FEPER

SISTEMUL DE OPERARE CP/M

IMPLEMENTAT PE MICROCALCULATOARELE

TPD SI JUNIOR

(II)

UZ INTERN

1986

Intreprinderea de Echipamente Periferice Bucuresti

SISTEMUL DE OPERARE CP/M

IMPLEMENTAT PE MICROCALCULATOARELE

TPD SI JUNIOR

(II)

MACROASAMBLOR
BASIC
FORTRAN
PASCAL
TURBO PASCAL
C

1986

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

ASAMBLORUL DE PROGRAME RELOCABILE M80

1986

CUPRINS

1. UTILIZAREA MACROASAMBLORULUI M80 2

2. LIMBAJUL SURSA ACCEPTAT DE MACROASAMBLORUL M80 4

3. DIRECTIVELE DE ASAMBLARE M80 6
3.1 Directive generale. 6
3.2 Directive si conventii pentru definirea de macroinstructiuni. 7
3.3 Directive de asamblare conditionata 8
3.4 Directive de asamblare repetata 9
3.5 Conventii utilizate in lista de asamblare 9

4. CODURI DE EROARE LA ASAMBLARE. 10

5. UTILIZAREA SUBPROGRAMELOR DIN BIBLIOTECA FORLIB.REL. 11

1. UTILIZAREA MACROASAMBLORULUI M80

Macroasamblorul M80 traduce programe sursa scrise in limbaj de asamblare 8080 sau Z80 intr-un format obiect relocabil, ce urmeaza a fi prelucrat de link-editorul L80.

Limbajul sursa acceptat contine un numar relativ mare de directive de asamblare, printre care directive pentru definire de macroinstructiuni, directive de asamblare conditionata, directive de control a listei de asamblare si altele.

<u>Comanda de asamblare:</u>

Sint posibile doua moduri de utilizare a asamblorului M80:

- pentru asamblarea unui singur program (modul) sursa se introduce o singura linie de comanda de forma:

```
M80 frel,fprn=fmac/opt1/opt2/...<CR>
```

- pentru asamblarea mai multor programe (module) sursa se incarca asamblorul prin comanda:

```
M80<CR>
```

si apoi se pot introduce mai multe linii de forma:

```
frel,fprn=fmac/opt1/opt2/...<CR>
```

ca raspuns la caracterul "*" ce arata ca asamblorul este gata sa primeasca o noua comanda.

Parametrii liniei de comanda au urmatoarea semnificatie:

- frel - fisier obiect relocabil produs de asamblor (are tipul implicit .REL);
- fprn - fisier de listare produs de asamblor (are tipul implicit .PRN, pe disc);
- fmac - fisier sursa citit de asamblor (are tipul implicit .MAC)

Numele de fisiere fmac,frel,fprn urmeaza formatul general :

```
d:nume.tip          sau          nume
```

Dispozitivul suport "d" poate lipsi cind se refera la discul implicit, iar tipul fisierelor poate lipsi daca se respecta conventiile implicite. Discul suport al fisierului de intrare este preluat implicit si pentru fisierele de iesire.

Optiunile de asamblare "opt" sint:

/C - genereaza referinte incrucisate pe lista de asamblare;
/L - produce fisier de listare;
/H - afisarea in hexa a memoriei pe lista de asamblare;
/O - afisarea in octal a memoriei pe lista de asamblare;
/R - genereaza fisier obiect;
/Z - genereaza cod obiect pentru Z80.

Ca fisier de listare se pot utiliza urmatoarele nume de dispozitive periferice:

TTY: pentru consola sistem;
LST: pentru imprimanta.

Fisierele frel si fprn sint optionale; de asemenea tipurile fisierelor sursa si obiect pot fi omise in comanda (daca fisierele sursa sint de tip .MAC).

La consola se afiseaza intotdeauna liniile cu erori , chiar daca nu se cere lista de asamblare.

Exemple de comenzi de asamblare:

1) M80 B:TESTM,TTY:=B:TESTM<CR>

Se asambleaza programul din fisierul B:TESTM.MAC, se genereaza fisierul B:TESTM.REL si se listeaza la consola codul sursa.

2) M80 =TEST.ASM<CR>

Se face o verificare sintactica a programului din fisierul TEST.ASM, fara a genera fisier obiect (erorile sint afisate).

3) M80 =B:TEST/R/L<CR>

Se asambleaza fisierul B:TEST.MAC si se produc doua fisiere pe acelasi disc: TEST.REL si TEST.PRN. Aceasta comanda este echivalenta cu comanda:

```
     M80 B:TEST,B:TEST=B:TEST<CR>
sau, mai complet:
     M80 B:TEST.REL,B:TEST.PRN=B:TEST.MAC<CR>
```

4)
```
M80<CR>
  *=P1/R<CR>
  *=P2/R<CR>
  *CTRL/C
```

Se asambleaza succesiv fisierele P1.MAC si P2.MAC generind fisierele P1.REL si P2.REL. Iesirea din asamblor se face cu CTRL/C.

2. LIMBAJUL SURSA ACCEPTAT DE ASAMBLORUL M80

Limbajul de asamblare acceptat de M80 este compatibil cu limbajele acceptate de alte asambloare pentru 8080, dar prezinta si unele particularitati ce sint prezentate in continuare.

Sintaxa programelor M80:

- O linie sursa poate avea max. 132 de caractere si, in fata, un numar de linie, cu conditia ca fiecare cifra din acest numar sa aiba bitul 7 setat.

- Literele mici sint acceptate numai in comentarii si in constante alfanumerice.

- In cadrul etichetelor simbolice sint admise urmatoarele caractere speciale, asimilate cu simbolurile: $, ., ?, @, -.

- Sint admise constante sau expresii in zona de cod mnemonic, ele fiind considerate ca operanzi ai unei directive DB implicite:

```
zero:    0
```

este echivalenta cu:

```
zero:    DB   0
```

- O eticheta urmata de doua caractere ":" este tratata ca simbol public (global):

```
SUBR::   PUSH     H
```

este echivalenta cu:

```
      PUBLIC   SUBR
          ...
      SUBR:    PUSH     H
```

- Un simbol urmat de doua caractere "#" este tratat ca simbol extern:

```
      CALL     MUL##
```

este echivalent cu:

```
      EXTRN    MUL
          ...
      CALL     MUL
```

- Constantele numerice pot fi binare (sufix B), octale (sufix O sau Q), hexazecimale (sufix H sau prefix X), zecimale (sufix D sau nici un sufix).

- Constantele numerice prea mari sint automat trunchiate la un cuvint (16 biti).

- Sirurile de caractere pot fi delimitate prin ghilimele simple sau duble.

- Expresiile din zona operand pot contine:
    - operatorii aritmetici +, -, *, / si semnul _;
    - operatorii simbolici NUL, LOW, HIGH, MOD, SHR, SHL;
    - operatorii de relatie EQ, NE, LT, LE, GT, GE;
    - operatorii logici NOT, AND, OR, XOR;
    - paranteze rotunde.

- Operatorii simbolici, de relatie si logici trebuie separati de operanzi prin spatii.

Se pot utiliza ca operanzi de un octet coduri simbolice de instructiuni:

```
MVI A,(JMP)
MVI C,MOV A,B
```

- Simbolurile sint de 4 tipuri:
  - absolute;
  - relative la segmentul de date;
  - relative la segmentul de program;
  - relative la blocul de comun;

Expresiile permise cu aceste simboluri sint:

<oarecare> + <absolut> -> <oarecare>
<oarecare> - <absolut> -> <oarecare>
<oarecare> - <oarecare> -> <absolut>

- Simbolurile externe pot fi folosite in expresii aritmetice cu operatorii +, -.

3. DIRECTIVELE ASAMBLORULUI M80

3.1 Directive generale

ASEG

Inceput de segment absolut. Acesta directiva este echivalenta cu directiva CSEG insotita de optiunea /P la link-editare.

```
COMMON    //
COMMON    /nume/
```

Inceput de bloc de date comun.

CSEG

Inceputul unui segment de cod. Directiva CSEG este implicita pentru orice programe, deci poate lipsi in general. Ea este necesara atunci cind segmentul de cod este precedat de un bloc comun, pentru delimitarea celor doua segmente.

```
DB   exp[,exp...]
DB   "sir"[,"sir"...]
```

Genereaza una sau mai multe constante de un octet, corespunzatoare expresiilor sau sirurilor de caractere din zona operand.

```
DC   "sir"
```

Genereaza un sir de caractere ASCII si pune 1 in bitul cel mai semnificativ (bitul 7) din ultimul caracter (pentru detectarea sfirsitului sirului de caractere)

```
DS   exp
```

Rezerva o zona de memorie de lungime egala cu valoarea expresiei "exp"

DSEG

Inceput de segment de date.

```
DW   exp[,exp...]
```

Genereaza una sau mai multe constante de un cuvint.

```
END  [exp]
```

Marcheaza sfirsitul unui program si, eventual indica adresa de lansare in executie.

```
ENTRY   nume [,nume...]
PUBLIC  nume  [,nume...]
```

Aceste doua directive echivalente declara unul sau mai multe simboluri publice (puncte de intrare), care pot fi folosite si din alte module asamblate separat.

```
nume    EQU  exp
```

Atribuie numelui din zona eticheta valoarea expresiei din zona operand.

```
EXTRN nume[,nume...]
EXT nume [,nume...]
```

Aceste doua directive echivalente declara unul sau mai multe simboluri externe, definite in alte module dar folosite in modulul curent.

```
NAME      ('nume')
```

Atribuie un nume modulului curent (echivalenta cu TITLE).

ORG exp
Stabileste adresa de implantare pentru codul sau zonele de date ce urmeaza.

PAGE [exp]
Produce trecerea la o pagina noua in cursul listei de asamblare. O pagina are implicit lungimea de 50 de linii sau lungimea data de operandul directivei (10-255).

nume SET exp

Ca si directiva EQU, atribuie numelui din stinga valoarea expresiei din dreapta dar permite redefinirea unui nume de mai multe ori in program.

SUBTTL "text"
Genereaza un subtitlu, de max. 60 de caractere, imprimat automat pe fiecare pagina de listare.

TITLE "text"
Genereaza un titlu, imprimat pe prima linie din fiecare pagina a listei de asamblare; primele 6 caractere se considera drept nume al modulului obiect generat.

.COMMEN /text/
Reprezinta o alta posibilitate de a scrie comentarii in textul sursa; ca delimitatori de inceput si de sfirsit comentariu se poate folosi orice caracter (nu neaparat caracterul "/").

.PRINTX /text/
Afiseaza la consola textul respectiv in momentul asamblarii acestei directive. Textul poate fi incadrat de orice caractere identice.

.RADIX exp
Modifica baza de numeratie implicita pentru constantele numerice.

.REQUEST fisier[,fisier...]
Cere linkeditorului sa caute in bibliotecile indicate pentru rezolvarea referintelor externe intilnite in program.

.XLIST / .LIST
Opreste/reia listarea textului sursa in fisierul de listare.

3.2 Directive si conventii pentru definirea de macroinstructiuni:

nume MACRO arg1,arg2,...
Inceput de macroinstructiune.

ENDM
Sfirsit de macroinstructiune.

EXITM
Termina expandarea unei instructiuni.

LOCAL arg1,arg2,...
Creaza cite un simbol unic pentru fiecare dintre argumentele formale continute in cursul expandarii unei macroinsstructiuni. Etichetele astfel generate au forma ..nnnn unde "n" este o cifra zecimala.

.LALL
Listeaza tot textul macro la toate expandarile.

.SALL
Listeaza numai codul obiect produs la o macroinstructiune.

.XALL
Interzice listarea macro-expandarilor.

Caracterul "&" se foloseste pentru concatenare de texte sau simboluri in cursul expandarii unei macroinstructiuni. Un argument formal introdus intre "" nu este substituit decit daca este precedat de "&".

Caracterul "!" utilizat inaintea unui caracter intr-un argument face ca acest caracter sa fie tratat ca un literal (sa fie copiat intocmai si nu substituit).

Parantezele ascutite ("< >") se folosesc tot pentru literali in cadrul argumentelor. Notatiile "!;" si "<;>" sint echivalente.
Comentariile dintr-o macro-definitie care sint precedate de ";;" nu sint memorate si deci nu mai apar la expandarea macroinstructiunii.

Directiva conditionala:

IF NUL <ARGUMENT>

are ca rezultat fals daca, in timpul expandarii, primul caracter din argument este orice altceva decit ";" sau <CR>.

3.3 Directive de asamblare conditionata

Directiva conditionala:

IF NUL arg
are un rezultat fals daca, in timpul expandarii, primul caracter din argument este orice altceva decit ";" sau <CR>.
Formatul general al unei asamblari conditionate este urmatorul:

```
IFxx  arg
  ...
[ELSE
  ...]
ENDIF
```

unde IFxx poate fi una dintre urmatoarele directive:

IF/IFT exp	-	adevarat daca <exp> este diferita de zero;
IFE/IFF exp	-	adevarat daca expresia <exp> este zero;
IF1	-	adevarat pentru prima trecere a asamblarii;
IF2	-	adevarat pentru a doua trecere a asamblarii;
IFNDEF simbol	-	adevarat daca <simbol> este nedefinit;
IFDEF simbol	-	adevarat daca simbol este definit sau este declarat extern;
IFB arg	-	adevarat daca argumentul este blanc (" ") (parantezele unghiulare sint necesare);
IFNB arg	-	adevarat daca argumentul este diferit de blanc.

Este permisa suprapunerea directivelor conditionale pe orice numar de nivele.
Argumentele directivelor conditionale trebuie sa fie cunoscute din prima trecere a asamblarii, deci sa fie definite inainte de a fi utilizate ca argumente.

3.4 Directive de asamblare repetata

REPT exp

Repeta instructiunile care urmeaza pina la o directiva ENDM de cite ori arata valoarea expresiei "exp" (valoare pe 16 biti).

IRP arg,<arg1,arg2,...>

Repeta instructiunile care urmeaza pina la o directiva ENDM de atitea ori cite argumente sint in lista dintre parantezele unghiulare, substituind de fiecare data argumentul formal cu argumentul urmator din lista.

Secventa urmatoare:

```
        IRP     X,<1,2,3,4>
        DB      X
        ENDM
```

este echivalenta cu secventa:

```
        X    SET  0
             REPT 4
        X    SET  X+1
        DB   X
        ENDM
```

si genereaza 4 directive DB:

```
        DB   1
        DB   2
        DB   3
        DB   4
```

IRPC arg,"sir"

Repeta instructiunile care urmeaza pina la o directiva ENDM de atitea ori cite caractere contine sirul dat, inlocuind de fiecare data argumentul formal cu caracterul urmator din sir.

3.5 Conventii utilizate in lista de asamblare

Formatul unei linii din lista de asamblare este urmatorul:

[crf] [err] loc m xx xxxx   linia sursa

unde:

crf - numar de referinta incrucisata (optional);
err - cod de eroare (o litera);
loc - adresa locatiei de memorie;
m - indicator mod de adresare;
xx... - continutul locatiei de memorie.

Indicatorul de mod "m" poate avea valorile urmatoare:

blank - adresa absoluta;
' - adresa relativa la segmentul de cod;
" - adresa relativa la segmentul de date;
! - adresa relativa in blocul comun;
* - referinta externa.

In tabela de simboli se folosesc urmatoarele notatii:

* - simbol extern;
i - simbol public (punct de intrare);
c - nume bloc comun;
u - simbol nedefinit.

4. CODURI DE EROARE LA ASAMBLARE

A (ARGUMENT ERROR)	- argument de directiva gresit;
C (CONDITIONAL NESTING ERROR)	- directive conditionale gresit suprapuse;
D (DOUBLE DEFINED SYMBOL)	- simbol dublu definit;
E (EXTERNAL ERROR)	- eroare la o referinta externa;
M (MULTIPLE DEFINED SYMBOL)	- simbol multiplu definit;
N (NUMBER ERROR)	- constanta numerica gresita;
O (OPCOD ERROR)	- cod numeric gresit sau alta eroare de sintaxa;
P (PHASE ERROR)	- eroare de faza la asamblare;
Q (QUESTIONABLE SYNTAX)	- linie terminata incorect;
R (RELOCATION ERROR)	- utilizare incorecta a unui simbol relocabil intr-o expresie;
U (UNDEFINED SYMBOL)	- simbol nedefinit;
V (VALUE ERROR)	- valoare gresita a unei constante;

5. UTILIZAREA SUBPROGRAMELOR DIN BIBLIOTECA FORLIB.REL

Biblioteca FORLIB.REL contine subprograme utilizate in mod normal de programele rezultate din compilari FORTRAN. O serie de subprograme din biblioteca sint de utilitate mai generala si pot fi apelate de programe scrise in limbaj de asamblare; aceste subprograme pot fi clasificate in trei categorii:

a) subrutine pentru operatii aritmetice;
b) subrutine pentru conversii de tip;
c) functii FORTRAN intrinseci (functii standard).

Conventiile de transmitere a argumentelor la aceste subprograme sint diferite de conventia de transmitere la apelarea de subprograme FORTRAN.

Subrutine pentru operatii aritmetice:

$AC = acumulator de virgula mobila in precizie simpla
($AC + 3 = adresa exponent);
$DAC = acumulator de virgula mobila in precizie dubla
($AC + 7 = adresa exponent).

Conventii:

a) argumentul 1 in registre:
- intregi in HL;
- reali in $AC;
- dubla-precizie in $DAC.

b) argumentul 2 in registre sau in memorie, functie de tip:
- intregi in HL sau DE daca HL contin primul argument;
- reali si dubla precizie in memorie, la adresa data de HL.

Functie	Nume	Arg.1	Arg.2
Adunare	$AA	R ($AC)	I (H,L)
	$AB	R ($AC)	R((H,L))
	$AQ	D ($DAC)	I (H,L)
	$AR	D ($DAC)	R((H,L))
	$AU	D ($DAC)	D((H,L))
Scadere	$SA	R ($AC)	I (H,L)
	$SB	R ($AC)	R((H,L))
	$SQ	D ($DAC)	I (H,L)
	$SR	D ($DAC)	R((H,L))
	$SU	D ($DAC)	D((H,L))
Inmultire	$M9	I (H,L)	I (D,E)
	$MA	R ($AC)	I (H,L)
	$MB	R ($AC)	R((H,L))
	$MQ	D ($DAC)	I (H,L)
	$MR	D ($DAC)	R((H,L))
	$MU	D ($DAC)	D((H,L))
Impartire	$D9	I (H,L)	I (D,E)
	$DA	R ($AC)	I (H,L)
	$DB	R ($AC)	R((H,L))
	$DQ	D ($DAC)	I (H,L)
	$DR	D ($DAC)	R((H,L))
	$DU	D ($DAC)	D((H,L))

Exponentiere	$ED	I (H,L)	I (D,E)
	$EA	R ($AC)	I (H,L)
	$EB	R ($AC)	R((H,L))
	$EQ	D ($DAC)	I (H,L)
	$ER	D ($DAC)	R((H,L))
	$EU	D ($DAC)	D((H,L))

Observatie:

La operatia de impartire intreaga ($D9) registrele HL trebuie sa contina impartitorul iar registrele DE trebuie sa contina deimpartitul. Dupa impartire, registrele HL contin citul intreg al impartirii, iar registrele DE contin restul impartirii intregi.

Subrutine pentru conversii de tip:

Argumente:
logic in A
intreg in HL
real in $AC
dubla-precizie in $DAC

Functiile de conversie sint date in urmatorul tabel:

Nume	Argument	Rezultat	Tip conversie
$CA	I (H,L)	R ($AC)	intreg-real
$CC	I (H,L)	D ($DAC)	intreg-dubla-precizie
$CH	R ($AC)	I (H,L)	real-intreg
$CJ	R ($AC)	L (A)	real-logic
$CK	R ($AC)	D ($DAC)	real-dubla precizie
$CX	D ($DAC)	I (H,L)	dubla precizie-intreg
$CY	D ($DAC)	R ($AC)	dubla precizie-real
$CZ	D ($DAC)	L (A)	dubla precizie-logic

In tabele s-au folosit urmatoarele notatii:

I = intreg;
R = real;
D = dubla precizie;
L = logic.

Functii FORTRAN intrinseci (functii standard):

Parametrii din HL si DE (al treilea argument in BC) reprezinta adresele argumentelor.

Pentru MIN si MAX numarul de argumente se transmite in A.

Cu exceptia functiilor INP si OUT, nici o functie nu admite argumente de un octet. Ele trebuie convertite, altfel apar rezultate impredictibile.

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

BASIC-80

1986

CUPRINS

1. INFORMATII GENERAE DESPRE BASIC-80 4
1.1 Mod de operare 4
1.2 Formatul liniei program 4
1.3 Set de caractere 4
1.4 Constante 5
1.5 Variabile 5
1.6 Expresii si operatori 6
1.7 Operatii asupra sirurilor de caractere 7
1.8 Caractere de control in BASIC 7

2. COMENZI SI INSTRUCTIUNI BASIC-80 9
2.1 AUTO 9
2.2 CALL 9
2.3 CHAIN 9
2.4 CLEAR 10
2.5 CLOSE 10
2.6 COMMON 10
2.7 CONT 10
2.8 DATA 11
2.9 DEF 11
2.10 DEFINT/SNG/DBL/STR 11
2.11 DEF USR 11
2.12 DELETE 12
2.13 DIM 12
2.14 EDIT 12
2.15 END 13
2.16 ERASE 13
2.17 ERR si ERL 13
2.18 ERROR 14
2.19 FIELD 14
2.20 FOR...NEXT 14
2.21 GET 15
2.22 GOSUB...RETURN 15
2.23 GOTO 15
2.24 IF...THEN[...ELSE] si IF...GOTO 15
2.25 INPUT 16
2.26 INPUT# 16
2.27 KILL 17
2.28 LET 17
2.29 LINE INPUT 17
2.30 LINE INPUT# 17
2.31 LIST 18
2.32 LLIST 18
2.33 LOAD 18
2.34 LPRINT si LPRINT USING 18
2.35 LSET si RSET 19
2.36 MERGE 19
2.37 MID$ 19
2.38 NAME 20
2.39 NEW 20
2.40 NULL 20
2.41 ON ERROR GOTO 20
2.42 ON GOSUB si ON GOTO 20
2.43 OPEN 21
2.44 OPTION BASE 21

2.45 OUT 21
2.46 POKE 22
2.47 PRINT 22
2.48 PRINT USING 22
2.49 PRINT# si PRINT# USING 24
2.50 PUT 24
2.51 RANDOMIZE 24
2.52 READ 25
2.53 REM 25
2.54 RENUM 25
2.55 RESTORE 26
2.56 RESUME 26
2.57 RUN 26
2.58 SAVE 26
2.59 STOP 27
2.60 SWAP 27
2.61 TRON/TROFF 27
2.62 WAIT 28
2.63 WHILE...WEND 28
2.64 WIDTH 28
2.65 WRITE 28
2.66 WRITE# 29

3. FUNCTII BASIC 30
3.1 ABS 30
3.2 ASC 30
3.3 ATN 30
3.4 CDBL 30
3.5 CHR$ 30
3.6 CINT 31
3.7 COS 31
3.8 CSNG 32
3.9 CVI, CVS, CVD 32
3.10 EOF 32
3.11 EXP 32
3.12 FIX 32
3.13 FRE 33
3.14 HEX$ 33
3.15 INKEY$ 33
3.16 INP 33
3.17 LEFT$ 33
3.18 INSTR 33
3.19 INT 34
3.20 LEFT$ 34
3.21 LEN 34
3.22 LOC 34
3.23 LOG 34
3.24 LPOS 35
3.25 MID$ 35
3.26 MKI$, MKS$, MKD$ 35
3.27 OCT$ 35
3.28 PEEK 35
3.29 POS 36
3.30 RIGHT$ 36
3.31 RND 36
3.32 SGN 36
3.33 SIN 36
3.34 SPACE$ 36
3.35 SPC 37
3.36 SQR 37

3.37 STR$ 37
3.38 STRING$ 37
3.39 TAB 37
3.40 TAN 37
3.41 USR 38
3.42 VAL 38
3.43 VARPTR 38

4. CREAREA SI EXPLOATAREA PROGRAMELOR FOLOSIND INTERPRETORUL BASIC-80 39
4.1 Functii 39
4.2 Instructiuni de salt 40
4.3 Instructiuni de ciclare 40
4.4 Instructiuni de intrare/iesire 41
4.5 Operatii de intrare/iesire pe disc 41
4.6 Executia unui program BASIC 41

5. CONVERSIA PROGRAMELOR IN BASIC-80 42

6. SUBRUTINE IN LIMBAJ DE ASAMBLARE APELABILE DIN BASIC 43
6.1 Alocarea memoriei 43
6.2 Instructiuni de apel 43
6.3 Lansarea BASIC in cazul folosirii subrutinelor in limbaj de asamblare ...44

7. EXEMPLE DE PROGRAME BASIC-80 45

8. COMENZI SUPLIMENTARE IN BASIC GRAFIC (GBASIC) 47
8.1 VIEWPORT 47
8.2 WINDOW 47
8.3 MOVE 47
8.4 RMOVE 47
8.5 DRAW 48
8.6 RDRAW 48
8.7 ROTATE 48
8.8 Exemplu de program grafic in BASIC 48

9. UTILIZAREA COMPILATORULUI BASIC 50
9.1 Forme de apelare 50
9.2 Optiuni de compilare 50

1. INFORMATII GENERALE DESPRE BASIC-80

Documentatia prezinta utilizarea interpretorului pentru limbajul BASIC-80 si a compilatorului BASIC-80, elaborate de fima Microsoft.

Limbajul acceptat este conform standardului ANSI din 1978.

Sint prezentate si comenzile din varianta grafica a interpretorului (comenzi grafice).

1.1 Mod de operare

Apelarea interpretorului BASIC se face cu comanda CP/M:

```
A>BASIC80<cr>
```

Reintoarcerea in sistemul CP/M se face cu comanda SYSTEM.

Dupa ce a fost lansat, BASIC-80 tipareste promptul "Ok" care arata ca interpretorul este gata sa accepte comenzi.

Interpretorul BASIC-80 poate fi utilizat in doua moduri: mod direct si mod indirect.

In mod direct instructiunile si comenzile nu sint precedate de un numar de linie si sint executate imediat ce sint introduse. Rezultatele operatiilor aritmetice si logice pot fi afisate imediat si stocate pentru o utilizare ultericara dar nu mai pot fi refolosite instructiunile de calcul.

Acest mod de lucru este util pentru depanare si la folosirea BASIC drept "calculator de buzunar" pentru operatii rapide care nu necesita un program complet.

Modul indirect este modul folosit pentru introducerea de programe. Liniile program sint precedate de un numar si sint stocate in memorie. Un program aflat in memorie este executat la introducerea comenzii RUN.

1.2 Formatul liniei program

O linie program poate avea urmatoarea forma:

```
NNNNN <instructiune BASIC> [ : <instructiune BASIC>...] <CR>
```

adica se pot introduce mai multe instructiuni pe o linie program separate prin ":". O linie program poate avea max. 255 caractere si se poate extinde pe mai multe linii fizice; in acest caz fiecare linie fizica este terminata cu <LF>.

Fiecare instructiune BASIC incepe cu un numar de linie (NNNNN). Acest numar arata ordinea in care liniile program vor fi stocate in memorie si este folosit ca referinta in instructiunile de salt si editare. Numarul liniei este cuprins intre 0 si 65529. Caracterul "." poate fi folosit in comenzile EDIT, LIST, AUTO si DELETE pentru a referi linia curenta.

1.3 Set de caractere

Setul de caractere cuprinde caractere alfabetice, caractere numerice si caractere speciale. Caracterele alfabetice reprezinta literele mari si mici ale alfabetului latin. Caractele numerice sint cifrele de la 0 la 9.

Caracterele speciale recunoscute de BASIC sint:

```
= + - * / ^ ( ) % # $ ! [ ] , ; : ' . & ? < > \ @ _
    <RUBOUT> <ESC> <LF> <CR>
```

BASIC recunoaste si urmatoarele caractere de control:

CTRL/A, CTRL/C, CTRL/G, CTRL/H, CTRL/I,
CTRL/O, CTRL/R, CTRL/S, CTRL/Q, CTRL/U.

1.4 Constante

Sint doua tipuri de constante: numerice si siruri.

O constanta de tip sir este o secventa de pina la 255 caractere alfanumerice cuprinse intre ghilimele. (Ex: "hello").

Observatie: numele unui fisier este considerat sir de caractere si trebuie incadrat de ghilimele la folosire.

Constantele numerice sint numere pozitive si negative. BASIC recunoaste 5 tipuri de constante numerice:

1. Constante intregi - cuprinse in intervalul -32768 +32767 (fara punct zecimal)
2. Constante virgula fixa. Ex.: 452.32. (contin un punct zecimal);
3. Constante virgula mobila (forma cu exponent). Ex.: 235E5.
4. Constante virgula mobila dubla precizie. Ex.: 235D5.
5. Constante hexazecimale - precedate de &H.
6. Constante octale - precedate de &O.

Constantele pot fi reprezentate in simpla precizie (memorate cu 7 cifre si reprezentate cu max. 6 cifre) si dubla precizie (memorate cu 16 cifre).

O constanta in precizie simpla este recunoscuta prin una din urmatoarele caracteristici:

- are max. 7 cifre zecimale;
- are forma cu exponent (cu litera E);
- se termina cu caracterul "!".

O constanta in dubla precizie este recunoscuta prin una din urmatoarele caracteristici:

- are peste 8 cifre zecimale;
- are o forma exponentiala (cu litera D);
- se termina cu caracterul #.

1.5 Variabile

In BASIC-80 variabilele pot sa aiba orice lungime dar numai primele 40 de caractere sint semnificative si reprezinta o valoare numerica sau un sir de caractere.

Variabilele pot fi variabile sir, intregi, simpla precizie sau dubla precizie.

Tipul unei variabile poate fi stabilit si printr-un caracter special ce urmeaza numele variabilei. Acest caracter poate fi:

$ - variabila de tip sir;
% - variabila intreaga;
! - variabila simpla precizie;
- variabila dubla precizie.

Este posibila si declararea explicita a tipului variabilelor prin instructiunile: DEFINT, DEFSTR, DEFSNG, DEFDBL.

Tipul implicit pentru o variabila este simpla precizie.

BASIC-80 accepta si variabile indexate. Numarul maxim pentru dimensiunile unei variabile indexate este 255. Numarul maxim de elemente/dimensiune este 32767.

O variabila indexata se declara folosind instructiunea DIM.

Numele unei variabile indexate este orice nume de variabila acceptat de BASIC si este urmat de una sau mai multe expresii intregi intre paranteze. De

exemplu: A(1,2), B(i,j), C(k).

Pentru diferitele variabile se rezerva spatiu astfel:

```
variabila simpla   - intreaga: 2 octeti;
                   - simpla precizie: 4 octeti;
                   - dubla precizie: 8 octeti;
variabila indexata - elemente intregi: 2 octeti/element;
                   - elemente simpla precizie: 4 octeti/element;
                   - elemente dubla precizie: 8 octeti/element;
variabila sir      - 3 octeti mai mult decit continutul sirului.
```

Numele unei variabile poate contine litere, cifre si caracterul punct (".") si trebuie sa nu coincida cu un nume rezervat; numele rezervate sint numele de comenzi, de instructiuni, de functii si de operatori (logici).

Daca un nume incepe cu literele FN atunci el este un nume de functie definita de utilizator.

Conversii de tip:

- daca se atribuie unei variabile numerice o constanta de un tip diferit de al variabilei, atunci valoarea numerica se converteste automat la tipul variabilei;
- la evaluarea expresiilor, toti operanzii sint convertiti automat la tipul operandului de precizie maxima, care este si tipul rezultatului expresiei;
- operatorii logici convertesc operanzii in intregi si produc un rezultat intreg;
- la conversia din virgula mobila in intreg, se face rotunjire la intregul cel mai apropiat si nu se trunchiaza partea fractionara;
- la conversia din dubla precizie in simpla precizie se retin numai primele 7 cifre zecimale, cu rotunjire la ultima cifra.

1.6 Expresii si operatori

O expresie poate fi o constanta, o variabila, sau o combinatie de constante si variabile legate prin operatori si eventual cuprinse intre paranteze rotunde.

Exista 4 categorii de operatori:

- aritmetici;
- relationali;
- logici;
- functionali.

1.6.1 Operatori aritmetici

BASIC permite ca operatii aritmetice adunarea (+), scaderea (-), inmultirea (*), impartirea (/), ridicarea la putere (^), impartirea intreaga (\) si restul impartirii intregi (MOD).

Exemple:

```
10 \ 4 = 2;
25.68 \ 6.99 = 3;
25.68 MOD 6.99 = 5.
```

Operatorii folositi sint cei indicati mai sus in paranteze. Prioritetea operatiilor este cea cunoscuta. Ordinea executarii operatiilor poate fi modificata folosind parantezele "(" si ")".

Operatorii aritmetici (\) si MOD se pot folosi si cu operanzi neintregi, care sint automat convertiti la intregi inainte de operatie.

In cazul unei depasiri sau impartiri cu 0 se afiseaza un mesaj, se pune ca rezultat numarul maxim reprezentabil si se continua executia programului.

1.6.2 Operatori relationali

Operatorii relationali permit compararea a doua valori si sint folositi pentru a compara expresii aritmetice sau siruri. Ei sint:

Simbol	Exemplu	Semnificatie
=	A = B	A este egal cu B
<	A < B	A mai mic decit B
>	A > B	A mai mare decit B
<= =<	A <= B	A mai mic sau egal cu B
>= =>	A => B	A mai mare sau egal cu B
<> ><	A <> B	A diferit de B

Rezultatul unei operatii relationale poate fi "adevarat" (valoare = -1) sau "fals" (valoare = 0).

Operatiile aritmetice sint evaluate inaintea operatiilor relationale.

1.6.3 Operatori logici

Rezultatul unei operatii logice este fie "adevarat" (valoare diferita de 0), fie "fals" (0).

Operatiile logice se fac bit cu bit.

Operatorii logici, in ordinea prioritatii lor, sint:

NOT - negatie;
AND - si logic;
OR - sau logic;
XOR - sau exclusiv;
IMP - implicatie (X IMP Y = 1 daca X <= Y);
EQV - echivalenta (X EQV Y = 1 daca X = Y).

1.6.4 Operatori functionali

In BASIC pot fi folosite doua tipuri de functii: functii intrinseci, predefinite si functii definite de utilizator (prin instructiunea DEFFN).

1.7 Operatii asupra sirurilor de caractere

Sirurile de caractere pot fi: concatenate (prin operatorul "+") sau comparate (prin operatorii de relatie).

1.8 Caractere de control in BASIC

CTRL/A - se trece in mod de lucru EDIT pe ultima linie introdusa;
CTRL/C - se intrerupe executia unui program si se da controlul interpretorului;
CTRL/G - genereaza un semnal sonor la terminal;
CTRL/H - sterge ultimul caracter tastat;
CTRL/I - tabulare fixata din 8 in 8 coloane;
CTRL/O - suspenda o asteptare de I/E si continua executia programului; urmatorul CTRL/O activeaza I/E;
CTRL/R - reafiseaza linia tastata;
CTRL/S - suspenda executia programului;

CTRL/Q - reia executia dupa o suspendare cu CTRL/S;
CTRL/U - sterge linia curenta.

2. COMENZI SI INSTRUCTIUNI BASIC-80

2.1 AUTO

Format:

```
AUTO [<nr.linie>[,<increment>]]
```

Efect: genereaza automat un numar de linie program dupa fiecare <CR>. Valorile implicite pentru <nr.linie> si <increment> sint 10 si 10.
Daca un numar de linie exista deja, dupa acesta apare "*" ca semn ca linia ce urmeaza va inlocui o linie existenta.
Iesirea din modul "AUTO" se face prin CTRL/C.

Exemplu:

```
AUTO 100,50 - genereaza numerele de linie 100,150,...;
AUTO        - genereaza numerele de linie 10,20,30...;
```

2.2 CALL

Format:

```
CALL <nume variabila>[<lista de argumente>]
```

Efect: se apeleaza o subrutina in limbaj de asamblare. <nume variabila> contine adresa de start a subrutinei, iar <lista de argumente> contine argumentele ce vor fi transmise subrutinei (poate contine numai variabile).
Instructiunea CALL genereaza aceiasi secventa de apelare ca in FORTRAN, COBOL, etc.

Exemplu:

```
10  MYROUT=&HD00
120 CALL MYROUT
```

2.3 CHAIN

Format:

```
CHAIN [MERGE] "<nume fisier>"[,[<exp.nr.linie>][,ALL]
        [,DELETE <multime>]]
```

Efect: apeleaza un program si ii transmite variabile din programul curent. "<nume fisier>" este numele programului apelat, <exp.nr.linie> este o expresie care da numarul liniei de start a programului apelat (implicit prima), "ALL" indica faptul ca toate variabilele programului curent vor fi transmise programului apelat (implicit sint transmise numai cele declarate cu COMMON).

Optiunea "MERGE" adauga programul apelat la programul din memorie, ca segment separat ce poate fi suprapus peste alt segment. Daca nu se foloseste optiunea "MERGE" atunci toate declaratiile de variabile din programul initial se pierd.

Optiunea "DELETE" sterge liniile specificate prin lista <multime> si se foloseste pentru stergerea unui segment inainte de a suprapune peste el un alt segment. (<multime> este de forma <nr.linie1>-<nr.linie2>). Numerele de linie din <multime> sint afectate de comanda RENUM.

2.4 CLEAR

Format:

```
CLEAR[,[<exp.1>][,<exp.2>]]
```

Efect: initializeaza toate variabilele numerice cu 0, toate variabilele de tip sir la valoarea "" (sir nul) si inchide toate fisierele deschise.

<exp.1> este o locatie de memorie care, daca este specificata, reprezinta limita superioara a memoriei permise pentru BASIC-80.

<exp.2> stabileste noul spatiu pentru stiva BASIC (dimensiunea implicita a stivei este de 256 de octeti).

Exemplu:

CLEAR	- toate variabilele iau valoarea 0 si se inchid toate fisierele;
CLEAR,32768	- fixeaza limita memoriei disponibile pentru interpretorul BASIC la adresa 32768;
CLEAR,,2000	- rezerva pentru stiva 2000 de octeti.

2.5 CLOSE

Format:

```
CLOSE[#]<nr.fisier>[,[#]<nr.fisier>...>]]
```

Efect: inchide un fisier pe disc. <nr.fisier> este numarul sub care fisierul a fost deschis. O instructiune CLOSE fara argumente inchide toate fisierele deschise.
Fisierul poate fi deschis sub acelasi numar sau sub alt numar.

Comenzile END si NEW inchid automat toate fisierele (STOP nu inchide fisiere).

2.6 COMMON

Format:

```
COMMON <lista de variabile>
```

Efect: transmite implicit variabilele din <lista variabile> unui program la executia unei instructiuni CHAIN. Variabilele de tip tablou trebuie sa fie insotite de paranteze (ex: COMMON A,B()). Este posibila si o instructiune COMMON similara cu cea din FORTRAN:

```
COMMON /<nume>/<lista de variabile>
```

pentru comunicare cu programe in FORTRAN sau in limbaj de asamblare, dar nu pentru transmiterea variabilelor la un program BASIC apelat cu CHAIN.

Exemplu:

```
100 COMMON A,B,C,D$
110 CHAIN "PROG3",10
```

2.7 CONT

Format:

CONT

Efect: continua executia unui program dupa ce aceasta a fost oprita de o comanda CTRL/C sau de o instructiune STOP sau END.
Se foloseste impreuna cu STOP pentru depanarea programelor sau pentru reluarea executiei dupa oprirea prin eroare.
CONT este invalid daca programul a fost schimbat prin editare dupa oprire.

2.8 DATA

Format:

DATA <lista constante>

Efect: memoreaza constante numerice si siruri de caractere care pot fi citite ulterior cu instructiunea READ.

Poate fi plasata oriunde in program, deoarece instructiunea READ acceseaza instructiunile DATA in ordine.

<lista constante> contine constante numerice in orice format.
La citire tipul variabilei trebuie sa corespunda cu cel stabilit prin DATA.
Citirea instructiunilor DATA poate fi reluata folosind instructiunea RESTORE.

2.9 DEF

Format:

DEF FN<nume>[<lista par.>]=<def.functie>

Efect: definirea unei functii scrise de utilizator.

<nume> trebuie sa fie o variabila legala.

<lista par.> contine variabilele folosite la definirea functiei (parametrii trebuie separati prin virgule), iar <def functie> este o expresie.

Definirea unei functii trebuie facuta inaintea utilizarii ei. O functie poate fi de tip numeric sau sir.

Definirea este ilegala in modul de lucru direct.

Exemplu:

```
410 DEF FNAB(x,y)=x^2/y^3
420 T=FNAB(i,j)
```

2.10 DEFINT/SNG/DBL/STR

Format:

DEF<tip><set de litere>

unde <tip> poate fi: INT, SNG, DBL sau STR.
Efect: declara tipul variabilelor ale caror nume incep cu una dintre literele specificate ca intreg, simpla precizie, dubla precizie sau sir. Implicit toate variabilele sint considerate de tip simpla precizie.

Exemplu:

```
DEFINT I-N : DEFSTR A
```

2.11 DEF USR

Format:

DEF USR[<cifra>]=<exp.intreaga>

Efect: specifica adresa de start a unei subrutine in limbaj de asamblare. <cifra> este intre 0 si 9 si corespunde numarului de subrutina a carei adresa este data. Daca este omis, rezulta ca se specifica adresa subrutinei 0 (DEF USR0).

Intr-un program pot apare mai multe DEF USR pentru a redefini punctul de start al unei subrutine, ceea ce permite accesul la un numar nelimitat de subrutine in limbaj de asamblare.

Exemplu:

```
200 DEF USR0 = 24000
210  X = USR0(X^2 / 2.89)
```

2.12 DELETE

Format:

DELETE [<nr.linie>][-<nr.linie>]

Efect: sterge linii din program. Daca linia de sters nu exista, apare un mesaj de eroare.

2.13 DIM

Format:

DIM<lista variabile>

Efect: specifica dimensiunile maxime ale variabilelor tablou. Daca dimensiunea unei variabile indexate nu este specificata, este stabilita implicit la 10.
Daca dimensiunea maxima este depasita apare un mesaj de eroare.
Instructiunea DIM stabileste ca valoare initiala a tuturor variabilelor din tablou zero.

Exemplu:

```
10 DIM A(20)
20 FOR I = 0 TO 20
30 READ A(I)
40 NEXT
```

2.14 EDIT

Format:

EDIT<nr.linie>

Efect: intra in modul de lucru editare la linia respectiva. Folosind EDIT se poate modifica o portiune dintr-o linie fara a reintroduce intreaga linie. Intrarea in mod editare pe linia curenta se poate face si cu CTRL/A.

<u>Subcomenzi de editare</u>

" " - muta cursorul la dreapta;

<RUBOUT>- muta cursorul la stinga;
I<text>$- insereaza text pe pozitia curenta a cursorului;
X - extinde linie. Muta cursorul la sfirsitul liniei unde se poate insera <text><CR> (este o inserare la sfirsit de linie);
nD - sterge n caractere la dreapta cursorului;
H - sterge toate caracterele de la dreapta cursorului;
nS<ch> - cauta a n-a aparitie a caracterului <ch> si pozitioneaza cursorul inainte de el;
nK<ch> - sterge a n-a aparitie a caracterului <ch>;
C<ch> - inlocuieste urmatorul caracter cu <ch>;
<CR> - se salveaza modificarile facute in linie si se iese din modul EDIT. Se tipareste restul liniei;
E - acelasi efect ca <CR>, insa fara tiparire;
Q - abandon editare linie;
L - listeaza restul liniei si pozitioneaza cursorul la inceputul liniei;
A - permite editarea unei linii din nou, refacind linia initiala si pozitionind cursorul pe inceput de linie.

2.15 END

Format:

```
END
```

Efect: termina executia programului, inchide toate fisierele si da controlul interpretorului BASIC-80.

Instructiunea END poate fi plasata oriunde in program pentru a termina executia. Spre deosebire de instructiunea STOP nu determina aparitia unui mesaj la consola.

2.16 ERASE

Format:

```
ERASE <lista de variabile tablou>
```

Efect: elimina variabilele tablou din program. Zona de memorie eliberata poate fi folosita pentru redimensionarea variabilelor tablou sau in alte scopuri.

2.17 ERR si ERL

Variabilele ERR si ERL sint folosite in caz de eroare.

Cind apare o eroare, ERR contine un cod de eroare si ERL numarul liniei program la care s-a produs eroarea. Pot fi folosite in instructiuni de tip IF.

Daca eroarea a aparut in modul de lucru direct, ERL contine 65535. Pentru a testa daca apare o erore in modul de lucru direct se foloseste:

```
IF 65535 = ERL THEN...
```

In alte cazuri se pot folosi formele:

```
IF ERR = <cod eroare> THEN...
IF ERL =  <nr.linie>  THEN...
```

ERL si ERR nu pot apare in instructiuni de atribuire deoarece sint variabile rezervate.

2.18 ERROR

Format:

ERROR <exp.intreaga>

Efect: 1. Simuleaza aparitia unei erori BASIC-80;
2. Permite definirea unor coduri de eroare de catre utilizator.
Valoarea expresiei (intre 0 si 255) reprezinta un cod de eroare BASIC.
La executia instructiunii ERROR apare mesajul de eroare corespunzator.
Pentru a defini un cod de eroare propriu, utilizatorul trebuie sa foloseasca o valoare mai mare decit cele folosite pentru coduri de eroare de BASIC-80. Acest cod de eroare utilizator trebuie sa duca la executia unei rutine de eroare.

Exemplu:

```
        ...
110 ON ERROR GOTO 400
120 INPUT "what is your bet";B
130 IF B>5000 THEN ERROR 210
        ...
400 IF ERR=210 THEN PRINT "house limit is $5000"
410 IF ERL=130 THEN RESUME 120
```

ERROR 15 <CR> - comanda tastata de utilizator in modul de lucru direct.

2.19 FIELD

Format:

FIELD[#]<nr.fisier>,<marime cimp>AS<sir variabile>

Efect: aloca spatiu pentru variabilele din <sir variabile> intr-un buffer de fisier in acces direct.

<nr.fisier> este numarul fixat pentru fisier la deschidere.

<marime cimp> este numarul de caractere ce vor fi alocate variabilelor din <sir variabile>.

Nu se foloseste pentru variabile utilizate in instructiuni LET sau INPUT.

Numarul total al octetilor alocati intr-o instructiune FIELD nu trebuie sa depaseasca lungimea inregistrarii specificate la deschiderea fisieruului in acces direct. Altfel apare o eroare de tipul "FIELD OVERFLOW" (lungimea implicita este de 128 octeti).

Se pot executa mai multe instructiuni FIELD pentru acelasi fisier. Executia unei instructiuni FIELD nu anuleaza efectul unei alte instructiuni FIELD.

In instructiunile INPUT si LET nu se utilizeaza nume de variabile folosite in instructiuni FIELD deoarece apar erori.

Exemplu:

```
FIELD 1 20 AS N$,10 AS ID$.
```

2.20 FOR...NEXT

Format:

```
FOR<variabila>=X TO Y [STEP Z]
    ...
NEXT [<variabila>][,,variabila>...]
```

unde X, Y si Z sint expresii numerice.

Efect: permite un ciclu de mai multe instructiuni.

<variabila> reprezinta contor pentru ciclu, X este valoarea sa initiala, Y valoarea sa finala, iar Z incrementul (implicit acesta este 1). Pasul Z poate fi si negativ.

Ciclurile FOR...NEXT pot fi suprapuse, dar trebuie sa aiba variabile de control diferite. O instructiune NEXT fara variabila este asociata cu ultima instructiune FOR.

2.21 GET

Format:

```
GET[#]<nr.fisier>[,<nr.intreg>]
```

Efect: citeste o inregistrare dintr-un fisier disc intr-un buffer in acces direct.

<nr. fisier> este numarul ce a fost atribuit fisierului la deschidere.

<nr.intreg> reprezinta numarul inregistrarii de citit. Daca este omis se citeste urmatoarea inregistrare din fisier. Cel mai mare numar de inregistrare posibil este 32767. Dupa o instructiune GET se pot citi caractere din buffer cu INPUT# sau LINE INPUT#.

2.22 GOSUB...RETURN

Format:

```
        GOSUB<nr.linie>
             ...
<nr.linie>   ...
        RETURN
```

Efect: apelul unei subrutine.

<nr.linie> este numarul primei linii din subrutina.

2.23 GOTO

Format:

```
GOTO<nr.linie>
```

Efect: salt neconditionat la linia cu numarul specificat, sau la prima instructiune executabila dupa linia specificata.

Exemplu:

```
10 READ R
20 PRINT "R = ";R
30 A = 3.14 * R ^ 2
40 PRINT "Area = ";A
50 GOTO 10
60 DATA 10,20,33
```

2.24 IF...THEN[...ELSE] si IF...GOTO

Format:

```
IF<exp.>THEN<instructiune>|<nr.linie>[ELSE<instructiune>|<nr.linie>]
```

sau:

```
IF<exp.>GOTO<nr.linie>[ELSE<instructiune>|<nr.linie>]
```

Efect: ia o decizie privind desfasurarea ulterioara a programului in functie de rezultatul evaluarii expresiei <exp.>:

daca <exp.> <> 0, se executa ramura lui THEN sau GOTO;

daca <exp.> = 0, se executa ramura lui ELSE daca exista.

Executia continua cu urmatoarea instructiune.

Este posibil ca o instructiune IF sa includa o alta instructiune IF.

Daca o instructiune IF...THEN este urmata de un numar de linie in modul de lucru direct, atunci apare o eroare de tipul "UNDEFINED LINE".

Cind se foloseste IF pentru a testa marimea rezultatului unei operatii cu numere in virgula mobila, trebuie sa se tina seama de faptul ca valoarea poate sa nu fie exacta. De exemplu, cind se doreste o valoare cit mai apropiata de 1.0 pentru o variabila A, testul va fi:

```
IF ABS(A - 1.0) < 1.0E-6 THEN...
```

Valoarea lui A este 1.0 cu o aproximatie de 1E-6.

Exemplu:

```
1) IF x > y THEN PRINT "Greater" ELSE IF y > x THEN PRINT "Less Than" ELSE
PRINT "Equal"
2) IF a = b THEN IF b = c THEN PRINT "a = c" ELSE PRINT "a <> c"
```

2.25 INPUT

Format:

```
INPUT[;][<"sir prompt">;]<lista variabile>
```

Efect: permite introducerea de date de la un terminal pe timpul executiei unui program.

Cind se intilneste o instructiune INPUT, executia programului se intrerupe si apare pe ecran un mesaj care anunta utilizatorul ca se asteapta date. Daca s-a introdus in instructiune si un <"sir prompt">, acesta este afisat inaintea cererii de date urmat de "?". Pentru a suprima caracterul "?" se poate folosi "," dupa prompt sau ";" dupa "INPUT".

Datele introduse sint asignate in ordine variabilelor din <lista variabile>.

Executia programului este reluata abia dupa ce s-au introdus valori pentru toate variabilele din lista.

Daca se raspunde cu mai multe sau mai putine date sau de tip necorespunzator (numeric in loc de sir, etc.) apare mesajul:

```
"?Redo from start"
```

Nici o asignare nu este facuta pina cind nu se da raspunsul corect.

Exemplu:

```
1) INPUT X (va afisa numai "?")
2) INPUT "data nasterii:",B$ (va afisa "data nasterii:")
```

2.26 INPUT#

Format:

```
INPUT#<nr.fisier>,<lista variabile>
```

Efect: citeste date dintr-un fisier secvential de pe disc si le atribuie variabilelor din program.

Primul caracter intilnit in fisier diferit de <CR>, spatiu sau <LF> reprezinta inceputul unui numar. Sfirsitul unui numar este considerat caracterul ce precede un spatiu, <CR>, sau ",".

Daca se intilneste caracterul ghilimele ("), se citeste un sir delimitat de alt caracter ghilimele sau de sfirsitul fisierului.

<nr.fisier> este numarul atribuit fisierului la deschidere.

<lista variabile> contine variabilele carora li se asigneaza valorile din fisier.

2.27 KILL

Format:

```
KILL"<nume fisier>"
```

Efect: se sterge fisierul specificat de pe disc.
Daca se incearca stergerea unui fisier deschis apare mesajul de eroare "file already open".
KILL poate fi utilizata pentru orice fisier de pe disc.

2.28 LET

Format:

```
[LET]<variabila>=<exp.>
```

Efect: se atribuie unei variabile rezultatul evaluarii expresiei <exp>. Cuvintul cheie LET poate lipsi.

2.29 LINE INPUT

Format:

```
LINE INPUT[;][<"sir prompt">;]<variabila sir>
```

Efect: citeste o linie terminata cu <CR> (pina la 255 caractere) intr-o variabila de tip sir de la consola.

<"sir prompt"> este un sir de caractere care, daca este prezent, este tiparit inainte ca intrarea sa fie acceptata.

Daca LINE INPUT este urmat de ";", un <CR> tastat de utilizator nu va avea ca ecou <CR><LF> la terminal, cum se intimpla cind lipseste acest caracter.

Iesirea din executia unei instructiuni LINE INPUT se face cu CTRL/C.

2.30 LINE INPUT#

Format:

```
LINE INPUT#<nr.fisier>,<variabila sir>
```

Efect: citeste o linie terminata cu <CR> (fara delimitatorii <CR><LF>), dintr-un fisier disc secvential intr-o variabila sir.

<nr.fisier> este numarul sub care a fost deschis fisierul, iar <variabila sir> este numele variabilei careia i se va asigna sirul de caractere din fisier

(pina la un <CR>).

2.31 LIST

Format:

```
LIST [<nr.linie>]
```

sau:

```
LIST [<nr.linie1>][-[<nr.linie2>]]]
```

Efect: listeaza o parte sau tot programul aflat in memorie la terminal.

Daca <nr.linie> este omis, programul este listat incepind cu linia cu cel mai mic numar si terminind cind se intilneste END (sau cind utilizatorul întrerupe listarea prin CTRL/C. Daca <nr.linie> este prezent se afiseaza la terminal numai linia cu acest numar.

Al doilea format permite urmatoarele optiuni:

- daca numai <nr.linie1> este specificat sint listate toate liniile program cu numarul mai mare sau egal cu acesta;
- daca se specifica numai <nr.linie2> se listeaza toate liniile de la inceputul programului pina la linia respectiva, inclusiv);
- daca ambele numere de linii sint specificate se listeaza portiunea de progam cuprinsa intre ele, inclusiv.

Exemplu:

```
1) LIST 100
2) LIST -200
3) LIST 100-200
```

2.32 LLIST

Format:

```
LLIST[<nr.linie1>[-[[<nr.linie2>]]]
```

Efect: listeaza o parte sau tot programul la imprimanta.
LLIST presupune existenta unei imprimante cu 132 caractere pe linie.

2.33 LOAD

Format:

```
LOAD"<nume fisier>"[,R]
```

Efect: incarca un fisier disc de tip .BAS in memorie.

LOAD inchide toate fisierele deschise anterior si sterge toate variabilele si liniile program aflate in memorie inaintea incarcarii.

Daca este prezenta optiunea "R", programul incarcat este si executat, iar toate fisierele de date deschise anterior sint pastrate neschimbate.

LOAD cu optiunea "R" poate fi folosit pentru a inlantui mai multe programe (sau segmente ale aceluiasi program).

Informatiile pot fi transmise intre diferitele programe (sau segmente) folosind fisierele de date pe disc.

2.34 LPRINT si LPRINT USING

Format:

```
LPRINT[<lista exp.>]
LPRINT USING <exp.sir>;<lista exp.>
```

Efect: tipareste valorile expresiilor din lista la imprimanta.

2.35 LSET si RSET

Format:

```
LSET<sir variabile>=<sir expresii>
RSET<sir variabile>=<sir expresii>
```

Efect: muta date din memorie intr-un buffer de fisier in acces direct (pentru pregatirea unei instructiuni PUT).

Daca sirul de expresii necesita mai putini octeti decit au fost rezervati cu FIELD pentru <sir variabile>, LSET aliniaza sirul la stinga iar RSET aliniaza sirul la dreapta. Restul cimpului se completeaza cu spatii.

Un sir prea lung de caractere este trunchiat la dreapta.

Valorile numerice trebuiesc convertite in siruri inainte de a fi transferate cu LSET sau RSET.

LSET sI RSET pot fi folosite si pentru alinierea unor siruri de caractere la stinga respectiv la dreapta.

Exemplu:

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

aliniaza la dreapta sirul N$ intr-un cimp de 20 de caractere.

2.36 MERGE

Format:

```
MERGE "<nume fisier>"
```

Efect: adauga fisierul disc specificat la programul curent din memorie.
Daca o linie din fisier are acelasi numar cu o linie din programul rezident in memorie, linia din fisier inlocuieste linia din memorie.

Dupa executarea unei instructiuni MERGE se intra in modul de lucru direct.

2.37 MID$

Format:

```
MID$(<expr.sir1>,N[,M])=<expr.sir2>
```

unde N si M sint expresii intregi si <expr.sir> este o expresie de tip sir (o variabila sau o constanta sir).
Efect: inlocuieste o portiune a unui sir cu alt sir.

Incepind cu pozitia N, caracterele din <expr.sir1> sint inlocuite cu cele din <expr.sir2> (sau cu M caractere din <expr.sir2> daca M este prezent).

Exemplu:

```
10 A$="KANSAS CITY, MO"
20 MID$(A$,14)="KS"
30 PRINT A$
```

2.38 NAME

Format:

NAME<nume vechi>=<nume nou>

Efect: schimba numele unui fisier disc.
<nume vechi> trebuie sa existe si <nume nou> sa nu fie prezent pe disc.
Exemplu:

```
NAME "accts" = "ledger"
```

Observatie:
O particularitate a interpretorului este faptul ca un nume de fisier scris cu caractere mici este considerat diferit de acelasi nume in care apar si caractere mari. Din aceasta cauza pot sa apara pe disc fisiere cu acelasi nume scrise cu caractere diferite. Deoarece CP/M nu poate face distinctie intre acestea, manipularea lor trebuie facuta numai in BASIC.

2.39 NEW

Format:

NEW

Efect: sterge programul si toate variabilele din memorie.
Comanda este folosita pentru a sterge memoria inaintea introducerii unui nou program.

2.40 NULL

Format:

NULL<expresie intreaga>

Efect: stabileste numarul de cifre 0 adaugate la sfirsitul fiecarei linii.

2.41 ON ERROR GOTO

Format:

ON ERROR GOTO<nr.linie>

Efect: salt in caz de eroare la rutina de tratare a erorii scrisa de utilizator.
O instructiune ON ERROR GOTO 0 sare la rutinele de tratare a erorilor BASIC-80 care tiparesc un mesaj de eroare si opresc executia.
Daca apare o eroare pe timpul executiei unei rutine de tratare a erorii, se afiseaza un mesaj de eroare BASIC si se opreste executia programului.

2.42 ON...GOSUB si ON...GOTO

Format:

ON<expresie>GOSUB<lista nr.linii>

ON<expresie>GOTO<lista nr.linii>

Efect: salt la una dintre diferitele linii specificate in lista, conditionat de valorea expresiei in momentul executarii instructiunii.
De exemplu, daca, in urma evaluarii, expresia ia valoarea 3, atunci saltul se va face la linia al carui numar ocupa pozitia 3 in lista (daca valoarea nu este intreaga se rotunjeste).

Daca valoarea expresiei este 0 sau mai mare decit numarul de elemente din lista, se continua cu urmatoarea linie executabila a programului.

Toate liniile care urmeaza dupa GOSUB trebuie sa fie linii de inceput de subrutine.

Daca valoarea <expresie> este negativa apare un mesaj de eroare.

2.43 OPEN

Format:

OPEN"<MOD>",[#]<nr.fisier>,"<nume fisier>",[<lung.inreg.>]

Efect: deschide un fisier pe disc.
Un fisier disc trebuie deschis inainte ca orice operatie I/E cu fisierul sa aiba loc. OPEN aloca un buffer pentru operatiile de I/E si determina modul de acces.

<MOD> este un sir de expresii in care primul caracter va fi unul dintre urmatoarele:

O - specifica mod de iesire secvential;
I - specifica mod de intrare secvential;
R - specifica mod I/E in acces direct.

<nr.fisier> este o expresie intreaga cu valori de la 1 la 15. Numarul este asociat fisierului pe toata durata de timp cit acesta este deschis si este folosit in instructiunile de I/E.

<lung.inreg.> este o expresie care, daca este prezenta, stabileste lungimea inregistrarii pentru operatii I/E in acces direct (implicit lungimea unei inregistrari este 128 octeti).

Un acelasi fisier poate fi deschis sub mai multe numere pentru citire secventiala sau acces direct, dar numai cu un singur numar pentru scriere secventiala.

Exemplu:

```
OPEN "R",2,"test"
```

2.44 OPTION BASE

Format:

OPTION BASE <N>

Efect: declara valoarea minima pentru indicii variabilelor indexate. Valoarea implicita este 0.

Exemplu:

```
OPTION BASE 1
```

Dupa executarea instructiunii cea mai mica valoare a unui indice de tablou folosit in program este 1.

2.45 OUT

Format:

OUT I,J

unde I si J sint expresii intregi.
Efect: transmite un octet la un port de iesire. I este numarul portului iar J octetul transmis.

2.46 POKE

Format:

POKE I,J

Efect: scrie octetul J in locatia de memorie I.
Functia complementara lui POKE este PEEK care citeste continutul unei locatii de memorie.
POKE si PEEK sint utile pentru memorarea eficienta a datelor, pentru incarcarea de subrutine in limbaj de asamblare, pentru transmiterea de argumente si rezultate la si de la subrutine in limbaj de asamblare.

2.47 PRINT

Format:

PRINT <lista expresii>

Efect: afiseaza date la terminal.
Daca <lista expresii> lipseste se afiseaza o linie goala.
Expresiile din lista pot fi numerice si/sau siruri de expresii.

<u>Pozitii la afisare</u>
Pozitia fiecarui element tiparit este determinata de caracterele folosite pentru separarea elemetelor in <lista expresii>. BASIC-80 imparte linia de afisare in zone de cite 14 pozitii fiecare. O virgula (",") intre doua elemente ale listei determina afisarea valorii urmatoare in urmatoarea zona. ";" determina afisarea valorii urmatoare imediat dupa cea anterioara. Unul sau mai multe spatii intre expresii au acelasi efect cu ";".
Daca se termina <lista expresii> cu "," sau ";" urmatoarea instructiune PRINT incepe afisarea in continuare. Altfel urmatoarea afisare incepe pe o linie nooua.
Daca se ajunge la sfirsitul unei linii fizice in timpul unei afisari se continua pe linia urmatoare.
Numerele afisate sint urmate de un spatiu.
Numerele pozitive sint precedate de un spatiu.

2.48 PRINT USING

Format:

PRINT USING <expresie sir>;<lista expresii>

Efect: afiseaza numere si/sau siruri folosind un format specific.
<lista expresii> este un sir de expresii sir sau numerice care vor fi afisate despartite prin ";".
<expresie sir> este un sir literal (sau variabil) ce cuprinde caractere speciale de format. Aceste caractere (prezentate mai jos) determina cimpurile de

forma numerelor sau sirurilor afisate.

Cind PRINT USING e folosita pentru a afisa siruri, unul din urmatoarele trei caractere pot fi utilizate pentru a determina formatul:

"!" - Se afiseaza numai primul caracter din sir;

"\n spatii\" - Se afiseaza 2+n caractere din sir. Daca n=0 se vor afisa 2 caractere din sir; daca sirul este mai lung decit cimpul rezervat pentru afisare, caracterele de la sfirsit sint ignorate; daca spatiul pentru afisare este mai mare decit sirul, acesta se va completa cu spatii la dreapta;

"&" - Specifica un cimp rezervat pentru afisarea unui sir de lungime variabila. Sirul este afisat in intregime, iar intre doua siruri nu apar spatii decit daca ele fac parte dintr-unul din siruri.

Daca se afiseaza numere, se folosesc urmatoarele caractere speciale:

"#" - Folosit pentru a reprezenta pozitia fiecarei cifre a numarului. Secventa de "#" trebuie sa fie urmata de numarul dorit de spatii. La reprezentarea numerelor zecimale cu acest format se vor face rotunjirile corespunzatoare.
Exemplu:

```
PRINT USING "##.##";5.3,6.789 produce 5.30 6.79;
```

"+" - Aflat la inceputul sau la sfirsitul sirului format determina afisarea semnelor numerelor (+ sau -) inaintea respectiv in urma lor;

"-" - Pus la sfirsitul formatului determina afisarea semnului pentru numere negative;

"**" - Puse la inceputul formatului determina aparitia unor asterisc-uri inaintea unor numere si ofera posibilitatea afisarii unor numere cu doua cifre mai lungi decit formatul specificat.
Exemplu:

```
PRINT USING "**#.# ";12.39,0.9,765.1 produce *12.4  **0.9  765.1;
```

"$$" - Duce la aparitia unui "$" imediat la stinga numarului. Creaza posibilitatea afisarii unei cifre in plus fata de format;

"**$" - Aflat la inceputul formatului combina efectele celor doua anterioare.
Exemplu:

```
PRINT USING "**$##.##";2.34 produce ***$2.34;
```

"," - O virgula aflata in format la stinga punctului zecimal determina aparitia unei virgule la fiecare 3 cifre in stinga punctului zecimal. O virgula aflata la sfirsitul formatului apare dupa fiecare numar afisat. Virgula nu are efect cind se utilizeaza formatul exponential (^^^^);

"^^^^" - Plasate dupa pozitiile cifrelor, specifica formatul exponential.
Exemplu:

```
PRINT USING "####^^^^";888888 produce .8889E06;
```

"\" - Urmatorul caracter din format se afiseaza.
Exemplu:

```
PRINT USING "\!##.##\!";11.22 produce !11.22!;
```

"%" - Daca numarul de afisat este mai mare (sau devine mai mare prin rotunjire) decit cimpul specificat prin format apare "%" inaintea numarului.

Exemplu:

```
PRINT USING ".##";.999 produce %1
PRINT USING "##.##";111.22 produce %111.22.
```

Daca numarul de cifre specificat este mai mare decit 24 apare un mesaj de eroare.

2.49 PRINT# si PRINT# USING

Format:

PRINT#<nr.fisier>[USING<expresie sir>;]<lista expresii>

Efect: scrie date intr-un fisier secvential.

<nr.fisier> este numarul folosit la deschiderea fisierului.

<expresie sir> este cea prezentata la instructiunea PRINT.

PRINT# nu compacteaza datele, ci le scrie asa cum ar fi afisate de PRINT.

Pentru a obtine un cimp compact de date in fisier se utilizeaza ca delimitator in <lista expresii> ";" (daca se foloseste "," se scriu pe disc si spatiile care ar apare la afisare).

Pentru a forma explicit sirurile pe disc, acestea trebuie explicit separate prin delimitatori.

Exemplu:

```
fie: A$="camera"
     B$="304"
     PRINT#1,A$;B$
va scrie pe disc: camera304
dar: PRINT#1,A$;",";B$
va scrie pe disc: camera,304
```

ceea ce poate fi citit in doua siruri identice cu cele de la care s-a pornit.

2.50 PUT

Format:

PUT[#]<nr.fisier>[,<nr.inreg.>]

Efect: scrie o inregistrare dintr-un buffer intr-un fisier in acces direct. Daca <nr.inreg.> lipseste atunci inregistrarea va primi numarul urmator celui dat ultimei inregistrari in fisier (la ultima folosire a instructiunii PUT). Cel mai mic numar de inregistrare este 1 si cel mai mare 32767.

Instructiunile PRINT#, PRINT# USING, WRITE# pot fi utilizate pentru a pune caractere intr-un buffer de fisier disc in acces direct inaintea unei instructiuni PUT.

In cazul instructiunii WRITE#, buffer-ul va fi umplut cu spatii dupa <CR>. Orice incercare de a citi sau scrie dupa sfirsitul buffer-ului duce la o eroare "field overflow".

2.51 RANDOMIZE

Format:

RANDOMIZE<expresie>

Efect: initializeaza generatorul de numere aleatoare.
Daca <expresie> lipseste, BASIC-80 suspenda asteapta o valoare numerica de la consola inainte de a executa RANDOMIZE.

Daca generatorul de numere aleatoare nu este resetat, functia RND va da aceeasi secventa de numere aleatoare la fiecare executie a unui program. Deci este bine sa se puna o instructiune RANDOMIZE la inceputul unui program ce foloseste numere aleatoare.

Exemplu:

```
10 RANDOMIZE
20 FOR I=1 TO 5
30 PRINT RND
40 NEXT I
```

2.52 READ

Format:

READ<lista variabile>

Efect: se se asigneaza variabilele cu valori din instructiuni DATA.
O instructiune READ poate folosi una sau mai multe instructiuni DATA si diferite instructiuni READ pot folosi aceeasi instructiune DATA, folosind instructiunea RESTORE.

Daca numarul variabilelor din <lista variabile> depaseste numarul de elemente dintr-o instructiume DATA, apare un mesaj "out of DATA".

Daca numarul de variabile specificat este mai mic decit numarul de elemente, urmatoarea instructiune READ va incepe citirea cu primul element necitit din instructiunea DATA.

Daca se termina instructiunile READ, datele necitite sint ignorate.

Pentru a reciti instructiuni DATA se foloseste RESTORE.

2.53 REM

Format:

REM <comentariu>

Efect: permite inserarea unor linii cu comentarii in program. Comentariile pot fi precedate de un apostrof oriunde in textul sursa.
Comentariile in instructiuni DATA vor fi considerate date valide.

Exemplu:

```
LET A=0 'initializare
```

2.54 RENUM

Format:

RENUM[[<nr.nou>[,<nr.vechi>][,<increment>]]]

Efect: renumerotarea liniilor program.

<nr.nou> este primul numar folosit in noua secventa (implicit este 10).

<nr.vechi> este numarul liniei cu care incepe renumerotarea (implicit prima linie program).

<increment> este valoarea cu care se mareste numarul de linie (daca lipseste se considera 10).

RENUM schimba corespunzator toate argumentele instructiunilor GOTO, GOSUB, THEN, etc.

RENUM nu poate fi folosita pentru a schimba ordinea liniilor in program.

2.55 RESTORE

Format:

RESTORE [<nr.linie>]

Efect: permite recitirea unei instructiuni DATA.
Daca <nr.linie> este prezent, prima instructiune READ care urmeaza instructiunii cu <nr.linie> specificat va citi prima instructiune DATA din program; altfel aceasta va fi citita de prima instructiune READ ce urmeaza lui RESTORE.

2.56 RESUME

Format:

RESUME
RESUME 0
RESUME NEXT
RESUME <nr.linie>

Efect: continua executia programului dupa o subrutina de tratare a erorii.
In cazul instructiunilor RESUME si RESUME 0 executia programului se reia de la instructiunea care a produs eroarea. Pentru RESUME NEXT, de la instructiunea urmatoare acesteia, iar pentru ultima forma de la linia specificata.

2.57 RUN

Format1:

RUN[<nr.linie>]

Efect: executa un program aflat in memorie (daca <nr.linie> exista, executia se incepe cu acea linie).

Format2:

RUN"<nume fisier>"[,R]

Efect: incarca un fisier program de pe disc si il executa.

RUN inchide toate fisierele deschise si sterge continutul memoriei inainte de a incarca un program de pe disc.

Daca optiunea "R" este prezenta, sint pastrate toate fisierele de date deschise anterior.

<nume fisier> este numele folosit la salvarea programului pe disc (cu instructiunea SAVE).

2.58 SAVE

Format:

SAVE "<nume fisier>"[,A/,P]

Efect: salveaza un program intr-un fisier disc.
Daca exista <nume fisier> pe disc, programul salvat va fi scris peste continutul vechiului fisier.

"A" determina salvarea programului in format ASCII. Daca lipseste, BASIC salveaza programul intr-un format ne-ASCII, mai eficient din punct de vedere al spatiului pe disc.

"P" duce la protejarea programului (orice incercare ulterioara de listare sau de editare a programului se va solda cu un mesaj de eroare).

Pentru a putea folosi ulterior instructiunea MERGE programul trebuie salvat in format ASCII.

2.59 STOP

Format:

```
STOP
```

Efect: incheie executia programului si reda controlul interpretorului BASIC.
Instructiunea STOP nu determina inchiderea fisierelor.
Daca se intilneste un STOP in program se afiseaza urmatorul mesaj:

```
Break in line nnnnn
```

2.60 SWAP

Format:

```
SWAP<variabila>,<variabila>
```

Efect: schimba intre ele valorile a doua variabile.
Cele doua variabile trebuie sa aiba acelasi tip (intreg, simpla precizie, dubla precizie, sir).

2.61 TRON/TROFF

Format:

```
TRON
TROFF
```

Efect: traseaza executia instructiunilor program in mod direct sau indirect. TRON determina tiparirea intre "[]" a numerelor liniilor executate. Trasarea unuia sau mai multor programe este intrerupta la intilnirea instructiunii TROFF.

Exemplu:

```
pentru secventa: 10 DATA 1,2,3
                 20 READ a,b,c
                 30 PRINT a,b,c
                 40 END
si comenzile:    TRON:
                  Ok
                 RUN
                 [10] [20] [30] 1 2 3
                 [40]
                  Ok
                 TROFF
```

2.62 WAIT

Format:

    WAIT <nr.port>,I,[J]

unde I si J sint expresii intregi.
Efect: suspenda executia programului pina cind portul specificat contine un anumit octet.
Data citita la portul <nr.port> intra intr-o expresie SAU LOGIC cu J (daca exista) si o expresie SI LOGIC cu I. Daca rezultatul este 0 portul este citit in ciclu. Altfel se executa urmatoarea instructiune.

2.63 WHILE...WEND

Format:

```
WHILE <expresie>
  ...
<instructiune>
  ...
WEND
```

Efect: se executa o serie de instructiuni in ciclu cit timp conditia este adevarata, adica expresia diferita de zero (ciclu cu testul la inceput).
Pot exista oricite nivele de cicluri de acest gen.
Fiecare WEND inchide cel mai apropiat WHILE.

2.64 WIDTH

Format:

    WIDTH [LPRINT] <expresie intreaga>

Efect: stabileste marimea liniei (in numar de caractere) pentru terminal sau pentru imprimanta (optiunea LPRINT).
<expresie intreaga> trebuie se fie intre 15 si 255.
Marimea implicita este de 72 de caractere.
Daca <expresie intreaga> are valoarea 255 marimea liniei este "infinita", adica interpretorul BASIC nu introduce nici un caracter <CR>.

2.65 WRITE

Format:

    WRITE [<lista de expresii>]

Efect: afiseaza date la terminal.
Daca <lista de expresii> lipseste, se afiseaza <CR><LF>.
Elementele listei trebuie separate prin ",".
Dupa ultimul element din lista, BASIC-80 insereaza <CR><LF>.
Exemplu:

```
10 a=80: b=90: c$="that's all"
20 WRITE a,b,c$
RUN
```

80,90, that's all

2.66 WRITE#

Format:

WRITE#<nr.fisier>,<lista expresii>

Efect: scrie date intr-un fisier secvential.

Diferenta dintre WRITE# si PRINT# consta in faptul ca WRITE# insereaza o virgula dupa fiecare element introdus in fisier si <CR> dupa ultimul element din lista.

Elementele listei trebuie departite prin ",".

<nr.fisier> este numarul sub care fisierul a fost deschis in modul "O". <lista expresii> este un sir de siruri sau expresii numerice ce trebuiesc separate prin virgula.

Dupa ultimul element din lista se insereaza <CR><LF> in fisier.

Exemplu:

```
10 LET a$="camera"
20 b$="93604-1"
30 WRITE# 1,a$,b$
```

3. FUNCTII BASIC

Aceste functii pot fi apelate in orice program fara a fi definite inainte. Argumentele functiilor sint de obicei inchise intre paranteze. In formatele date pentru functii in acest paragraf argumentele vor avea urmatoarele semnificatii:

x , y - orice expresii numerice;
i , j - expresii de numere intregi;
x$,y$ - expresii sir.

Daca se foloseste un numar zecimal in locul unui numar intreg, numarul zecimal va fi rotunjit.

Functiile interpretoarelor BASIC-80 si BASIC-86 au ca rezultate numai valori intregi si in simpla precizie.

Numai functiile compilatorului BASIC poate oferi rezultate in dubla precizie.

3.1 ABS

Format:

```
ABS(x)
```

Intoarce valoarea absoluta a expresiei x.

3.2 ASC

Format:

```
ASC(x$)
```

Intoarce codul primului caracter din sirul x$. Daca x$ este nul se da un mesaj de eroare.

3.3 ATN

Format:

```
ATN(x)
```

Intoarce valoarea arctg(x) in radiani.
Rezultatul este cuprins intre -pi/2 si pi/2 si este in simpla precizie.

3.4 CDBL

Format:

```
CDBL(x)
```

Converteste un numar x in dubla precizie.

3.5 CHR$

Format:

CHR$(i)

Intoarce caracterul cu codul ASCII "i".

Pentru microcalculatoarele TPD si JUNIOR comanda poate fi folosita si pentru a trimite caractere de control la ecranul consolei.

Secventa de stabilire a regimului de afisare a caracterelor este urmatoarea:

1BH, 56H, cod de comanda

Codurile de comanda pentru regimurile de lucru cu ecranul, pentru microcalculatoarele TPD si JUNIOR sint:

Cod hexa	Cod zecimal	Regim de lucru
80	128	normal de lucru
81	129	luminos
82	130	clipitor
83	131	clipitor + luminos
90	144	video invers
91	145	video invers + luminos
92	146	video invers + clipitor
93	147	video invers + luminos + clipitor
A0	160	subliniat
A1	161	subliniat + luminos
A2	162	subliniat + clipitor
A3	163	subliniat + clipitor + luminos
B0	176	subliniat + video invers
B1	177	subliniat + video invers + luminos
B2	178	subliniat + video invers + clipitor
B3	179	subliniat + video invers + clipitor + luminos

Exemplu:

```
PRINT CHR$(27);CHR$(86);CHR$(144);"test";CHR$(27);CHR$(86);CHR$(128)
```

se va afisa pe video invers cuvintul "test". Acelasi lucru se poate obtine folosind codurile de comanda in hexazecimal:

```
PRINT CHR$(&H1B);CHR$(&H56);CHR$(&H90);"test";CHR$(&H1B);CHR$(H56);CHR$(&H80)
```

Din exemple se observa ca revenirea la un regim normal de afisare se face folosind primul cod de comanda din tabel.

3.6 CINT

Format:

CINT(x)

Converteste numarul x la un numar intreg prin rotunjirea partii fractionare.

3.7 COS

Format:

COS(x)

Intoarce valoarea cos(x) unde x este in radiani.

3.8 CSNG

Format:

CSNG(x)

Converteste pe x intr-un numar in simpla precizie.

3.9 CVI, CVS, CVD

Format:

CVI(<sir de 2 oct.>)
CVS(<sir de 4 oct.>)
CVD(<sir de 8 oct.>)

Converteste un sir de octeti in valoare numerica.

Sirurile numerice citite dintr-un fisier in acces direct trebuie convertite in valori numerice inaintea folosirii lor in expresii aritmetice, etc.

CVI converteste un sir de 2 caractere la un intreg, CVS un sir de 4 caractere la o valoare in simpla precizie, iar CVD un sir de 8 octeti la o valoare in dubla precizie.

3.10 EOF

Format:

EOF("<nume fisier>")

Intoarce -1 (true) daca s-a atins sfirsitul unui fisier secvential.

Este bine sa se foloseasca EOF la citirea datelor dintr-un fisier pentru a evita aparitia unei erori la incercarea de a citi date dupa ce s-a atins sfirsitul de fisier.

3.11 EXP

Format:

EXP(x)

Intoarce valoarea exponentialei.

Daca x este mai mare decit 87.3365 apare un mesaj de depasire si executia programului continua.

3.12 FIX

Format:

FIX(x)

Trunchiaza numarul x si intoarce partea sa intrega.

3.13 FRE

Format:

```
FRE(0)
FRE(x$)
```

Folosind aceasta functie se poate obtine numarul octetilor de memorie RAM nefolositi inca de BASIC-80.

Argumentul functiei este un parametru formal.

FRE(" ") activeaza o "colectare" a octetilor nefolositi inainte de a da numarul lor. Colectarea dureaza 1 pina la 1,5 minute.

3.14 HEX$

Format:

```
HEX$(x)
```

Intoarce valoarea hexazecimala a unui numar intreg.

Daca x nu este numar intreg atunci el va fi rotunjit inainte de a se calcula valoarea hexazecimala.

3.15 INKEY$

Format:

```
INKEY$
```

Intoarce un caracter citit de la consolă sau un sir nul daca nu a fost introdus nici un caracter de la terminal.
Caracterele primite de INKEY nu sint afisate in ecou.

3.16 INP

Format:

```
INP(i)
```

Intoarce octetul citit la portul i (i trebuie sa fie in gama 0-255).
INP este functia complementara instructiunii OUT.

3.17 INPUT$

Format:

```
INPUT$(x[,[#]y])
```

Intoarce un sir de x caractere citite de la terminal sau din fisierul y (daca y este specificat).
Caracterele citite nu vor fi afisate.

3.18 INSTR

Format:

INSTR([i,]x$,y$)

Cauta prima aparitie a sirului y$ in sirul x$ si intoarce pozitia de la care incepe corespondenta. Deplasamentul i (optional) indica pozitia de la care se incepe cautarea.

Daca i > LEN(x$) sau x$ este nul sau y$ nu poate fi regasit in x$ functia intoarce valoarea 0.

Daca y$ este sir nul functia are ca rezultat i (daca exista), sau 1.

3.19 INT

Format:

INT(x)

Intoarce partea intreaga a lui x.

3.20 LEFT$

Format:

LEFT$(x$,i)

Are ca rezultat un subsir al lui x$ cuprins intre pozitiile 1 si i inclusiv. Daca i < LEN(x$) se obtine intregul sir; daca i = 0 se obtine sirul nul.

3.21 LEN

Format:

LEN(x$)

Intoarce lungimea sirului x$.
Sint numarate si caracterele netiparibile si spatiile.

3.22 LOC

Format:

LOC(<nr.fisier>)

Intoarce numarul ultimei inregistrari citite sau scrise pentru fisier disc deschis in acces direct.

Daca fisierul a fost deschis dar nu a avut loc nici o operatie de I/E, LOC intoarce valoarea 0.

Pentru fisierele disc secventiale LOC intoarce numarul de sectoare care au fost citite sau scrise.

3.23 LOG

Format:

LOG(x)

Intoarce logaritmul natural al lui x. Trebuie indeplinita conditia x > 0.

3.24 LPOS

Format:

    LPOS(x)

Intoarce pozitia urmatorului caracter tiparibil din buffer. x este un parametru formal.

3.25 MID$

Format:

    MID$(x$,i[,j])

Intoarce un subsir de j caractere din x$, care incep cu pozitia i.

Daca j este omis sau j > LEN(x$)-i, atunci se obtine subsirul de lungime LEN(x$)-i+1. Daca i > LEN(x$), MID$ intoarce un sir nul.

Daca i = 0, apare un mesaj de eroare.

3.26 MKI$, MKS$, MKD$

Format:

    MKI$(<expresie intreaga>)
    MKS$(<exp.simpla precizie>)
    MKD$(<exp.dubla precizie>)

Aceste functii convertesc valori numerice in siruri de valori.

Orice valoare plasata intr-un buffer in acces direct cu instructiunile LSET sau RSET trebuie convertita in sir de valori.

MKI$ converteste un intreg intr-un sir de 2 octeti, MKS$ converteste un numar simpla precizie intr-un sir de 4 octeti, iar MKD$ un numar dubla precizie intr-un sir de 8 octeti.

3.27 OCT$

Format:

    OCT$(x)

Intoarce valoarea octala a unui numar intreg. x este rotunjit la valoare intreaga inaintea conversiei.

3.28 PEEK

Format:

    PEEK(i)

Intoarce valoarea intreaga a octetului citit din memorie in locatia i.
PEEK este functia complementara lui POKE.

3.29 POS

Format:

POS(i)

Intoarce pozitia curenta a cursorului pe linie.
Cea mai din stinga pozitie este considerata 1.

3.30 RIGHT$

Format:

RIGHT$(x$,i)

Intoarce un subsir al lui x$ ce cuprinde caracterele de la inceputul lui x$ piná la pozitia i.

Daca i >= LEN(x$) atunci se obtine tot sirul x$. Daca i = 0 se obtine un sir nul.

3.31 RND

Format:

RND[(x)]

Intoarce o valoare aleatoare in intervalul [0,1].

Pe parcursul a mai multor executii ale aceluiasi program se va obtine aceeasi secventa de numere aleatoare daca generatorul nu este resetat (v. RANDOMIZE).

x < 0 duce la generarea aceleiasi secvente pentru orice x.
x > 0 sau lipsa lui x va duce la generarea urmatorului numar din secventa.
x = 0 repeta ultimul numar aleator generat.

3.32 SGN

Format:

SGN(x)

Daca x > 0 SGN(x) = 1;
Daca x = 0 SGN(x) = 0;
Daca x < 0 SGN(x) = -1.

3.33 SIN

Format:

SIN(x)

Are ca rezultat sin(x) calculat in simpla precizie. x este in radiani.

3.34 SPACE$

Format:

    SPACE$(x)

Intoarce un sir de spatii de lungime x. (x este rotunjit si trebuie sa fie cuprins in gama 0-255).

3.35 SPC

Format:

    SPC(i)

Se afiseaza i spatii la terminal.
SPC poate fi folosit numai in instructiuni PRINT si LPRINT.

3.36 SQR

Format:

    SQR(x)

Intoarce radacina patrata din x.
Trebuie indeplinita conditia x >= 0.

3.37 STR$

Format:

    STR$(x)

Intoarce un sir de cifre care reprezinta valoarea x.

3.38 STRING$

Format:

    STRING$(i,j)
    STRING$(i,x$)

Intoarce un sir de lungime i, in care toate caracterele au codul ASCII j sau codul primului caracter din sirul x$.

3.39 TAB

Format:

    TAB(i)

Introduce spatii pe linia afisata pina in pozitia i.
Daca pozitia curenta de afisare a depasit pozitia i, TAB introduce spatii pina la pozitia i de pe linia urmatoare.

3.40 TAN

Format:

TAN(x)

Intoarce valoarea tangentei lui x. x este in radiani, iar valoarea TAN(x) este data in simpla precizie.

3.41 USR

Format:

USR[<cifra>](x)

Apeleaza o subrutina scrisa de utilizator in limbaj de asamblare.
<cifra> apartine intervalului 0-9 si este aceeasi cu cea din instructiunea DEFUSR corespunzatoare subrutinei.
Implicit <cifra> = 0.

3.42 VAL

Format:

VAL(x$)

Intoarce valoarea numerica a sirului x$. VAl ignora spatiile, <LF> si <CR> din sirul argument.

3.43 VARPTR

Format1:

VARPTR(<nume variabila>)

Format2:

VARPTR(#<nr.fisier>)

1. Intoarce adresa primului octet al datei identificate cu <nume variabila>.
Trebuie atribuita o valoare pentru <nume variabila> inaintea folosirii VARPTR, altfel apare o eroare.
Valoarea obtinuta este cuprinsa intre -32768 si 32767. Daca se obtine o valoare negativa, aceasta va fi adunata la 65536 pentru a obtine adresa reala.
Este folosita pentru a transmite adresa unei variabile unei subrutine de asamblare.

2. Pentru fisiere secventiale, intoarce adresa de inceput a buffer-ului de I/E asignat pentru <nr.fisier>. Pentru fisierele deschise in acces direct intoarce adresa "buffer field" asignat pentru <nr.fisier>.

4. CREAREA SI EXPLOATAREA PROGRAMELOR FOLOSIND INTERPRETORUL BASIC-80

Crearea unui program BASIC-80 se face in modul de lucru indirect, in care fiecare linie program este precedata de un numar de linie de maximum 5 cifre. Formatul liniei program este cel aratat in 1.2.

Comentariile se introduc in program precedate de "REM" sau incadrate de (vezi 2.53).

Stergerea unei linii se face tastind numarul liniei urmat de <CR> sau folosind comanda DELETE (2.12).

Inlocuirea unei linii se face prin introducerea unei linii cu acelasi numar.

Corectia unei linii se face folosind comanda EDIT (2.14).

Afisarea la consola a programului se obtine cu comanda LIST (2.31), iar copia lui la imprimanta cu LLIST (2.32).

Numerotarea automata a liniilor program este executata dupa o comanda AUTO (2.1), iar renumerotarea dupa RENUM (2.54).

Atribuirea de valori variabilelor se face folosind instructiunea LET (2.28). Expresiile folosite la atribuire pot contine constante, expresii aritmetice sau functii.

4.1 Functii

Functiile folosite in expresii pot fi cele implicite sau pot fi definite de utilizator.

<u>Functii aritmetice</u>

ABS(x) - valoarea absoluta a lui x;
INT(x) - partea intreaga a lui x;
SQR(x) - radical din x;
EXP(x) - exponentiala de x;
LOG(x) - logaritm natural de x;
RND(x) - numar aleator intre 0 si 1;
SGN(x) - semnul lui x;
COS(x) - cosinus de x;
SIN(x) - sinus de x;
TAG(x) - tangenta de x;
ATN(x) - arctangenta de x.

<u>Functiile care opereaza cu valori numerice si au ca rezultate siruri:</u>

CHR$(x)	- genereaza un caracter cu codul ASCII i;
HEX$(x)	- genereaza valoarea hexazecimala a lui x;
OCT$(x)	- genereaza valoarea octala a lui x;
STR$(x)	- genereaza reprezentarea ASCII a valorii x;
SPC(i)	- afiseaza i spatii la terminal;
SPACE$(x)	- genereaza un sir de spatii de lungime x;
STRING$(i,j)	- genereaza un sir de lungime i in care toate caracterele au codul ASCII j;
STRING$(i,x$)	- genereaza un sir de i caractere identice cu primul caracter din sirul x$;
TAB(i)	- afiseaza spatii pe linie pina la pozitia i;
MKI$, MKS$, MKD$	- convertesc valori numerice in sirurile de cifre corespunzatoare.

Functiile care opereaza cu siruri si au ca rezultat valori numerice:

ASC(x$) - intoarce codul ASCII al primului caracter din sirul x$;
LEN(x$) - intoarce lungimea sirului x$;
INSTR([i],x$,y$) - cauta prima aparitie a sirului y$ in sirul x$ (incepind cu pozitia i) si intoarce pozitia de la care incepe coincidenta.

Functiile care opereaza cu siruri si au ca rezultat siruri:

LEFT$(x$,i) - intoarce subsirul cuprins intre pozitiile 1 si i in x$;
RIGHT(x$,i) - intoarce subsirul cuprins intre pozitia i si sfirsitul sirului x$;
MID$(x$,[i,j]) - intoarce un subsir de j caractere incepind cu pozitia i.

Functii de conversie numerica:

CDBL(x) - conversia lui x in dubla precizie;
CINT(x) - intoarce partea intreaga a lui x;
CSNG(x) - conversia lui x in simpla precizie;
CVI(x) \
CVS(x) - conversii de siruri de cifre in valori numerice.
CVD(x) /

Functii de intrare:

INKEY$ - intoarce caracterul tastat la consola fara afisare in ecou;
INPUT$(x[,[#]y]) - intoarce un sir de x caractere citite de la consola;
INP(i) - intoarce octetul citit de la portul i;
POS(i) - intoarce pozitia curenta a cursorului pe linie;
LPOS(i) - intoarce pozitia curenta a urmatorului caracter tiparibil;
PEEK(i) - intoarce octetul citit din locatia de memorie i.

Alte functii disponibile:

FREE - intoarce numarul octetilor liberi din memorie;
USR - apel de subrutina in limbaj de asamblare;
VARPTR - intoarce adresa primului octet al datei;
LOC - intoarce numarul ultimei inregistrari citite sau scrise intr-un fisier in acces direct;
EOF - arata daca s-a atins sfirsitul de fisier.

4.2 Instructiuni de salt

Instructiunile de salt recunoscute de interpretor sint:

GOTO <nr.linie> - salt neconditionat la linia indicata;
IF...THEN...[ELSE] \
IF...GOTO (10.2.24.) - instructiuni de salt conditionat;
ON...GOTO (10.2.42.) /
ON ERROR GOTO - salt la secventa de tratare a erorii.

4.3 Instructiuni de ciclare

Instructiunile de ciclare permise sint:

```
FOR <variabila> = x TO y [STEP z]
<instructiuni>
```

```
NEXT [<variabila>][,<variabila>...]
si
WHILE <expresie>
<instructiuni>
WEND
```

4.4 Instructiuni de intrare/iesire

Citirea unui caracter tastat la consola se face cu instructiunea INPUT, iar a unei linii cu LINE INPUT.

Afisarea unui sir de caractere sau a unor valori se face cu instructiunile PRINT si PRINT USING.

Tiparirea la imprimanta a unor date se face cu instructiunile LPRINT, LPRINT USING si WRITE.

Marimea unei linii afisate sau tiparite se poate schimba folosind WIDTH.

Citirea unor constante inscrise in memorie folosind instructiunea DATA se face folosind instructiunea READ. Reluarea citirii unor zone de date este facuta dupa executia unei instructiuni RESTORE.

Cu instructiunea OUT se poate transmite un octet la un port.

Cu instructiunea POKE se obtine inscrierea in memorie a octetului dorit.

4.5 Operatii de intrare/iesire pe disc

SAVE	- salvarea unei zone program intr-un fisier pe disc;
LOAD	- incarcarea unui program in memorie dintr-un fisier;
OPEN	- deschiderea unui fisier de date pe disc pentru operatii de citire / scriere;
INPUT#, LINE INPUT#	- citire de date dintr-un fisier disc secvential;
PRINT#, PRINT USING#, WRITE	- scriere de date intr-un fisier disc secvential;
GET	- citeste o inregistrare dintr-un fisier intr-un buffer in acces direct;
PUT	- scrie o inregistrare dintr-un buffer intr-un fisier in acces direct;
KILL	- sterge fisierul dorit de pe disc;
NAME	- schimba numele unui fisier disc.

4.6 Executia unui program BASIC

Execuutia unui program BASIC este inceputa dupa o comanda RUN si sfi.sita la intilnirea unui END sau STOP in program.

La testarea unui program se pot folosi instructiunile TRON / TROFF care determina afisarea numerelor liniilor executate.

Folosirea unei instructiuni WAIT duce la suspendarea executiei programului pina cind portul specificat contine un octet.

Se pot adauga la programul existent in memorie instructiuni continute intr-un fisier disc folosind MERGE.

Din programul rezident se poate apela un program de pe disc cu instructiunea CHAIN. Programul rezident poate transmite variabile programului apelat.

Oprirea fortata a executiei unui program si reintorcerea sub controlul interpretorului BASIC-80 se realizeaza prin tastarea CTRL/C. Continuarea unui program oprit fortat incepe dupa o comanda CONT.

Initializarea generatorului de numere aleatoare se face cu RANDOMIZE.

Interpretorul BASIC-80 are posibilitatea detectarii si tratarii erorilor program (ERR, ERL, ERROR).

Reluarea executiei dupa tratarea unei erori se face dupa o instructiune RESUME.

5. CONVERSIA PROGRAMELOR IN BASIC-80

Pentru a folosi programele scrise pentru alte interpretoare sau compilatoare sub controlul lui BASIC-80:

- se sterg toate instructiunile folosite pentru a declara lungimea sirurilor;
- daca se folosesc "," sau "&" pentru concatenare de siruri, acestea vor fi inlocuite cu "+";
- trebuie schimbate forme ca a$(i) pentru al i-lea caracter din sir sau a$(i,j);
- atribuiri de forma: LET b=c=0 se schimba in: c=0 : b=0;
- daca separatorul de instructiuni aflate pe aceeasi linie este altul decit ":" se va schimba;
- daca programul foloseste functii MAT (matrici) trebuie rescris folosind cicluri FOR.

6. SUBRUTINE IN LIMBAJ DE ASAMBLARE APELABILE DIN BASIC

6.1 Alocarea memoriei

Spatiul de memorie necesar unei subrutine in limbaj de asamblare trebuie rezervat de la inceputul lucrului cu interpretorul BASIC. In momentul apelarii interpretorului se introduce adresa limita a zonei de memorie necesare subrutinelor scrise in limbaj de asamblare cu switch-ul "/M:".

BASIC-80 foloseste toata memoria permisa de la punctul sau de start pina la ultima locatie care se afla in afara zonei afectate subrutinelor in limbaj de asamblare.

Subrutinele in limbaj de asamblare pot fi incarcate in memorie folosind comenzi ale sistemului de operare, cu instructiunea POKE, sau asamblate cu macroasamblorul M80 si incarcate cu linkeditorul L80.

6.2 Instructiuni de apel

Pentru apel se poate folosi comanda USR:

USR[<cifra>](<argument>)

unde <cifra> este cuprinsa intre 0 si 9 si <argument> este orice expresie numerica sau sir. <cifra> specifica ce subrutina a fost apelata (vezi DEF USR).

Cind se face un apel folosind USR, registrul A contine o valoare ce specifica tipul argumentului dat:

2 - un intreg pe doi octeti (in complement fata de 2);
3 - un sir;
4 - un numar simpla precizie in virgula mobila;
8 - un numar dubla precizie, virgula mobila.

Daca argumentul este un numar, in registrele HL se afla pointer-ul catre zona unde argumentul este stocat (numita FAC).

Daca argumentul este un intreg, FAC-3 contine octetul inferior si FAC-2 contine octetul superior al numarului.

Daca argumentul este un numar simpla precizie virgula mobila, FAC-3 contine octetul inferior al mantisei, FAC-2 octetul urmator al mantisei, iar FAC-1 cei mai semnificativi 8 biti. Bitul 7 al celui mai semnificativ octet este bit de semn (0 = pozitiv, 1 = negativ). FAC contine exponentul - 128. Punctul zecimal este la stinga celui mai semnificativ bit al mantisei.

Daca argumentul este un numar in virgula mobila dubla precizie, FAC-7 pina la FAC-4 contin inca 4 octeti ai mantisei.

Daca argumentul este un sir, DE dau referinta catre 3 octeti numiti "descriptor de sir". Octetul 0 contine lungimea sirului (0-255), iar octetii 1 si 2 contin adresa de inceput a sirului.

Daca argumentul este un sir literal din program, descriptorul poate da referinta catre textul programului. Acest lucru trebuie facut cu grija pentru a nu distruge programul. Pentru a evita aparitia unor rezultate imprevizibile se adauga un " " la sfirsitul sirului din program.

Exemplu:

```
a$="BASIC"+" "
```

Apelul functiilor utilizator scrise in limbaj de asamblare poate fi facut si cu instructiunea CALL ca in cazul compilatoarelor FORTRAN, COBOL si BASIC:

1. Daca numarul parametrilor este mai mic sau egal cu 3, acestia sint transferati prin registre.
2. Daca numarul parametrilor este mai mare ca 3, transferul se face dupa cum

urmeaza:
- parametrul 1 in HL;
- parametrul 2 in DE;
- parametrul 3,..., parametrul n intr-un bloc continuu de memorie indicat de registrele BC (BC indica adresa octetului cel mai putin semnificativ din parametrul 3).

6.3 Lansarea BASIC in cazul folosirii subrutinelor in limbaj de asamblare

A>BASIC [<nf>] [/F:<nrf>] [/M:<lsm>] [/S:<lmi>]

unde,

<nf> - <nume fisier>;
<nrf> - <numar fisiere>;
<lsm> - <limita superioara a memoriei>;
<lmi> - <lungimea maxima a inregistrarii>.

Daca <nume fisier> este prezent, BASIC-80 procedeaza ca si cum s-ar tasta "RUN <nume fisier>" dupa lansarea interpretorului.

Daca optiunea /F este omisa, numarul implicit de fisiere de date este 3. Fiecare fisier de date alocat de /F necesita 166 octeti de memorie.

Daca optiunea /S este omisa, marimea inregistrarii va fi de 128 octeti.

Daca optiunea /M este omisa, interpretorul foloseste toata memoria pina la punctul de start BDOS.

7. EXEMPLE DE PROGRAME BASIC-80

1.Crearea unui fisier de date secvential:

```
10 OPEN "o",#1,"date"
20 INPUT "nume";n$
30 IF n$=" " THEN END
40 INPUT "adresa";a$
50 INPUT "data nasterii";b$
60 PRINT#1,n$;",";d$;",";b$
70 PRINT:GOTO 20
```

2.Accesul la un fisier secvential de date:

```
10 OPEN "i",#1,"date"
15 IF EOF(1)=-1 THEN END
20 INPUT#1,n$,a$,b$
30 RIGHT$(b$,2)="78" THEN PRINT n$
40 GOTO 20
```

3.Completarea fisierului secvential de date:

```
10 ON ERROR GOTO 2000
20 OPEN "i",#1,"date"
30 ' daca fisierul exista, se citeste pentru a fi copiat
40 OPEN "o",#2,"copie"
50 IF EOF(1)=-1 THEN 90
60 LINE INPUT#,a$
70 PRINT#2,a$
80 GOTO 50
90 CLOSE #1
100 KILL "date"
110 REM se adauga noi date la fisier
120 INPUT "nume";n$
130 IF n$=" " THEN 200
140 LINE INPUT "adresa";a$
150 LINE INPUT "data nasterii";b$
160 PRINT#2,n$
170 PRINT#2,a$
180 PRINT#2,b$
190 PRINT:GOTO 120
200 CLOSE
205 ' schimba numele fisierului
210 NAME "copie" AS "date"
2000 IF ERR=53 AND ERL=20 THEN OPEN "o",#2,"copie" : RESUME 120
2010 ON ERROR GOTO 0
```

4.Crearea unui fisier de date in acces direct:

```
10 OPEN "r",#1,"date",42
20 FIELD #1,20 AS n$,14 AS a$,8 AS b$
30 INPUT "nr.inregistrare";cod%
40 INPUT "nume";x$
50 INPUT "adresa";s$
60 INPUT "data nasterii"
70 LSET n$=x$
80 PUT #1,cod%
110 GOTO 30
```

5.Citirea unui fisier in acces direct:

```
10 OPEN "r",#1,"date"42
20 FIELD #1,20 AS n$,14 AS a$,8 AS b$
30 INPUT "nr.inregistrare";cod%
40 GET#1,code%
50 PRINT n$
60 PRINT USING "$$###.##";CVS(a$)
70 PRINT b$:PRINT
80 GOTO 30
```

8. COMENZI SUPLIMENTARE IN BASIC GRAFIC (GBASIC)

Varianta grafica a interpretorului BASIC pentru TPD grafic (GBASIC) este diferita de cea pentru Felix M118 dar admite aceleasi comenzi.

8.1 VIEWPORT

Format:

```
VIEWPORT x1,x2,y1,y2
```

Efect: stabileste coordonatele zonei imagine (numita si fereastra ecran). Fereastra ecran este un dreptunghi cu coltul din stinga jos in punctul de coordonate (x1,y1) si cu coltul din dreapta sus in punctul (x2,y2). Coordonatele x1,x2,y1,y2 sint coordonate ecran exprimate in numar de puncte :

0 <= x1 < x2 < 511
0 <= y1 < y2 < 255

Pe parcursul unui program se poate redefini fereastra ecran, dar la un moment dat exista o singura fereastra ecran stabilita prin ultima comanda VIEWPORT.

8.2 WINDOW

Format:

```
WINDOW x1,x2,y1,y2
```

Efect: stabileste coordonatele ferestrei virtuale (numita si fereastra de date). Fereastra virtuala defineste domeniul datelor folosite in reprezentarile grafice; vor fi desenate pe ecran numai punctele cu abscisa intre x1 si x2 si ordonata intre y1 si y2.
Coordonatele ferestrei de date pot avea orice valori intregi sau reale.
Punctele din fereastra virtuala se proiecteaza pe ecran in zona imagine definita cu VIEWPORT printr-o scalare si o translatie.
Descrierea imaginilor desenate se face in coordonate virtuale din cadrul ferestrei virtuale.

8.3 MOVE

Format:

```
MOVE x,y
```

Efect: muta spotul in punctul de coordonate virtuale absolute (x,y).

8.4 RMOVE

Format:

```
RMOVE dx,dy
```

Efect: muta spotul in punctul cu coordonate virtuale relative (dx,dy) fata de punctul curent (punctul de plecare).

8.5 DRAW

Format:

DRAW x,y

Efect: deseneaza o dreapta din punctul curent pina in punctul cu coord. virtuale absolute (x,y).

8.6 RDRAW

Format:

RDRAW dx,dy

Efect: deseneaza o dreapta din punctul curent pina in punctul cu coord. virtuale relative (dx,dy) fata de punctul curent.

8.7 ROTATE

Format:

ROTATE u

Efect: rotatie cu unghiul "u" a liniilor desenate in coordonate relative. Unghiul "u" este exprimat in radiani; valori pozitive indica o rotatie in sens trigonometric direct (invers acelor de ceas).
Efectul comenzii ROTATE ramine valabil pina la urmatoarea comanda ROTATE. Readucerea in starea initiala se face cu ROTATE 0.

Pentru desenare este necesar ca ecranul sa fie in mod "pagina". Aducerea in mod pagina si stergerea ecranului se face cu secvente de Escape interpretate de sistem:

```
stergere : PRINT CHR$(27)+"I"
schimba mod : PRINT CHR$(27)+"2"
```

8.8 Exemplu de program grafic in BASIC

```
01   REM Rotirea unui patrat in jurul unui colt
10   PRINT CHR$(27)+"I" : PRINT CHR$(27)+"2"
20   WINDOW 0,100,0,100
30   VIEWPORT 20,104,0,100
40   FOR K=1 TO 20
50   INPUT N 'numar de rotatii
60   PRINT CHR$(24) 'sterge ecran
70   MOVE 50,50
80   GOSUB 200 'desenare patrat
90   FOR I=1 TO N
100  ROTATE 6.28/N*I
110  GOSUB 200
120  NEXT
130  NEXT
150  REM subrutina desenare patrat
200  RDRAW 0,40
210  RDRAW 40,0
220  RDRAW 0,-40
230  RDRAW -40,0
```

```
250  RETURN
```

9. UTILIZAREA COMPILATORULUI BASIC

Capitolul descriei modul de utilizare al compilatorului BASIC-80 livrat de firma Microsoft sub numele de BASCOM (5.30 din 1981).

Fisierele necesare pentru compilare BASIC sint :

```
BASCOM.COM - Compilator BASIC
BASLIB.REL - Biblioteca de module relocabile ptr BASIC
BRUN.ACOM  - Fisier necesar la executia programelor BASIC
L80.COM    - Linkeditor ,versiunea 3.44
```

Utilizarea compilatorului BASCOM este asemanatoare cu utilizarea compilatorului Fortran F80.

9.1 Forme de apelare

```
1) BASCOM frel,fprn=fbas/opt

2) BASCOM
   *frel,fprn=fbas/opt
```

Ambele forme sint echivalente si realizeaza o singura compilare.
"frel" este fisierul obiect de tip .REL creat de compilator.
"fprn" este fisierul listing de tip .PRN creat de compilator.
"fbas" este fisierul sursa de tip .BAS citit de compilator.
"opt" optiuni de compilare.
Oricare dintre aceste fisiere poate lipsi din linia de comanda.
Ca fisiere de listare se pot utiliza si dispozitivele TTY:,LST:.

9.2 Optiuni de compilare

/N nu se listeaza codul masina generat de compilator
/R genereaza fisier de tip .REL
/L genereaza fisier de tip .PRN

Exemple:

```
1) BASCOM ,TTY:=TEST/N
2) BASCOM TEST=TEST
3) BASCOM TEST,TEST=TEST/N
4) BASCOM =TEST/R/L
5) BASCOM
   *=TEST
```

Fisierele relocabile create de BASCOM trebuie legate impreuna cu biblioteca BASLIB folosind versiunea 3.44 a linkeditorului L80.

Exemple:

```
1) L80 TEST,BASLIB/S/G
2) L80 TEST,BASLIB/S,TEST/N/E
```

Pentru executarea unui program provenit din compilare BASIC este necesar sa existe si fisierul BRUN.COM pe acelasi disc cu fisierul de tip .COM creat de L80.

Fisierele sursa pentru compilatorul BASCOM au acelasi format ca si pentru

interpretorul BASIC ,dar fiecare linie sursa trebuie precedata de un numar de linie. Aceste fisiere pot fi create fie cu un editor de texte fie cu interpretorul si salvate in format ASCII (SAVE "nume",A).

Limbajul BASIC acceptat de compilatorul BASCOM este in general acelasi cu limbajul acceptat de interpretor, cu urmatoarele diferente:

- Nu sint permise comenzile specifice modului interactiv: AUTO, CONT, DELETE, EDIT, LIST, LLIST, LOAD, MERGE, NEW, RENUM, SAVE.
- Exista o serie de functii predefinite noi, in special pentru numere reale in dubla precizie.

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

COMPILATORUL FORTRAN F80

1986

CUPRINS

1. OPERARE 2

2. PARTICULARITATI ALE COMPILATORULUI FORTRAN F80. 4

3. FUNCTII DE BIBLIOTECA 9

4. MESAJE DE EROARE LA EXECUTIA PROGRAMELOR FORTRAN. 11

5. EXEMPLE DE PROGRAME FORTRAN 12

1. OPERARE

Documentatia prezentata se refera la compilatorul FORTRAN F80 elaborat de firma Microsoft, versiunea 3.41.

Compilatorul F80 accepta la intrare programe scrise in limbajul FORTRAN-IV (standard ANSI 1966) pe care le traduce in module obiect relocabile, ce urmeaza a fi prelucrate de linkeditorul L80. La editarea de legaturi este necesara biblioteca FORTRAN numita FORLIB.REL. Modulele rezultate din compilarea FORTRAN pot fi legate cu module obiect rezultate din asamblare cu macro-asamblorul M80.

Sint posibile doua moduri de utilizare a compilatorului F80, dupa cum se compileaza unul sau mai multe fisiere de tip .FOR:

- pentru o singura compilare:

```
F80 frel,fprn=ffor/opt1/opt2/...<CR>
```

- pentru mai multe compilari succesive:

```
F80<CR>
*frel,fprn=ffor/opt1/opt2...<CR>
*frel,fprn=ffor/opt1/opt2...<CR>
*^C<CR>
```

Iesirea din ciclul de compilari succesive se face cu CTRL/C.

Fisierele de intrare si de iesire la compilare sint:

frel = fisier obiect relocabil rezultat din compilare (are tipul implicit .REL);
fprn = fisier de listare a compilarii (pe disc are tipul implicit .PRN). Ca fisiere de listare pot fi utilizate si numele de dispozitive TTY: si LST:;
ffor = fisier sursa cu programul FORTRAN (tipul implicit .FOR)

Numele fisierelor frel, fprn, ffor au forma generala: d:nume.tip, cu mentiunea ca discul suport "d" si/sau tipul pot lipsi. Compilatorul considera in mod implicit ca fisierele de iesire frel si fprn se creaza pe acelasi disc unde se afla si fisierul de intrare ffor.

Optiunile posibile la o compilare FORTRAN (opt1,opt2,...) sint:

/L - se cere crearea unui fisier de listare (daca acest fisier nu apare explicit in linia de comanda);
/H - se cere afisarea in hexazecimal a adreselor de memorie (optiune implicita);
/M - se genereaza cod pentru a fi executat dintr-o memorie ROM.
/N - se suprima listarea codului generat de compilator in forma simbolica (in mod normal lista de compilare contine codul masina generat de compilator), precum si informatii despre alocarea memoriei pentru date si instructiuni.
/O - se cere afisarea in octal a adreselor de memorie;
/P - se cere alocarea a inca 100 de octeti pentru stiva utilizața de compilator (daca se constata o depasire a stivei la compilare);
/R - se cere generarea unui fisier obiect relocabil (daca acest fisier nu apare in mod explicit in linia de comanda).

Exemple de comenzi de compilare:

1) F80 TESTF,LST:=TESTF<CR>

Se compileaza programul din fisierul TESTF.FOR, se genereaza fisierul obiect TESTF.REL si se afiseaza la imprimanta lista de compilare.

2) F80 ,TTY:=B:TESTF<CR>

Se face o verificare sintactica a programului din fisierul TESTF.FOR, cu afisare la consola a intregului program, fara a se genera cod obiect.

3) F80 =B:TESTF/L/R<CR>

Aceasta comanda este echivalenta cu comanda:

F80 B:TESTF,B:TESTF=B:TESTF<CR>

si cu comanda:

F80 B:TESTF.REL,TESTF.PRN=B:TESTF.FOR<CR>

4) F80<CR>
*=FORT1/R<CR>
*=FORT2/R<CR>
*^C

2. PARTICULARITATI ALE COMPILATORULUI FORTRAN F80 (diferente fata de standardul FORTRAN ANSI 1966)

Limbajul acceptat de F80 are urmatoarele particularitati fata de standardul FORTRAN ANSI 1966:

- Nu exista tipul de date complex.

- Tipurile de date diferite se reprezinta pe lungimi diferite:
  - tipul logic pe un octet (8 biti);
  - tipul intreg pe 2 octeti (16 biti);
  - tipul real pe 4 octeti (32 biti);
  - tipul dubla-precizie pe 8 octeti (64 biti).

- In caz de continuare a unei instructiuni DO pe doua linii trebuie ca semnul de atribuire ("=") si prima virgula din instructiune sa apara pe prima linie.

- In lista de intrare/iesire a unei instructiuni READ sau WRITE nu sint permise subliste in paranteze.

- Declaratiile neexecutabile trebuie sa fie scrise la inceputul unui program in ordinea urmatoare:

```
PROGRAM,SUBROUTINE,FUNCTION,BLOCK DATA
IMPLICIT
DIMENSION,INTEGER,REAL,LOGICAL,DOUBLE PRECISION,EXTERNAL
COMMON
EQUIVALENCE
DATA
FUNCTII-INSTRUCTIUNE
```

  Un program principal FORTRAN poate fi precedat, optional, de instructiunea:
      PROGRAM nume
  Daca instructiunea lipseste programul primeste automat numele "$MAIN".

- O instructiune se poate continua pe orice numar de linii.

- Se pot folosi constante hexazecimale de forma X'HHHH' sau Z'HHHH' oriunde sint permise constante intregi (daca sint mai putin de 4 cifre hexazecimale, atunci valoarea constantei se aliniaza la dreapta pe 16 biti si se completeaza in stinga cu zerouri.

- In expresii se pot utiliza constante alfanumericg (unul sau doua caractere intre ghilimele) oriunde sint permise constante intregi (caracterele sint interpretate ca numar intreg).

- Ca argument efectiv la apelarea de proceduri sau functii se pot utiliza literali alfanumerici de orice lungime (siruri de caractere intre ghilimele simple).

- Sint permise expresii mixte si atribuiri mixte, iar conversiile de tip se realizeaza automat.

- Intr-o instructiune FORMAT, in instructiunea DATA sau in expresii un text constant se poate scrie intre ghilimele sau precedat de "nH"; deci forma 'text' este echivalenta cu nHtext.

- Instructiunile READ si WRITE pot utiliza optiunile "ERR=n" si "END=n",

pentru salt automat la o secventa de tratare a erorilor si, respectiv, la o secventa de tratare a sfirsitului de fisier.

- Functiile tip instructiune pot utiliza variabile indexate ca argumente formale.

- In instructiunile STOP c si PAUSE c, constanta "c" poate fi formata din 1-6 caractere ASCII (intre ghilimele).

- Exista urmatoarele subprograme standard suplimentare:
PEEK(a) functie logica ce are ca rezultat continutul locatiei de memorie cu adresa a;
POKE(a1,a2) depune la adresa a1 continutul adresei a2;
INP(p) functie logica ce are ca rezultat continutul portului de I/E cu numarul p;
OUT(p,a) subrutina care trimite la portul p de I/E continutul adresei de memorie a.

- Compilatorul nu face distinctie intre litere mari si litere mici, dar se recomanda utilizarea de litere mari.

- Reprezentarea datelor:
  - constantele logice au urmatoarea reprezentare hexazecimala:.FALSE.= 00,.TRUE.= FF sau orice valoare nenula;
  - variabilele logice se pot utiliza si ca variabile intregi de un octet in expresii aritmetice sau in comparatii;
  - variabilele intregi pot avea valori intre -32768 si +32767;
  - variabilele reale au o precizie de 7 cifre zecimale si un exponent cuprins intre -38 si +38;
  - variabilele reale cu precizie dubla au 16 cifre zecimale exacte si acelasi exponent maxim ca variabilele reale simpla precizie;

- Sint permise urmatoarele instructiuni pentru declararea tipului datelor:
```
INTEGER, REAL, LOGICAL, DOUBLE PRECISION
BYTE, INTEGER*1, INTEGER*2, REAL*4, REAL*8
```
Urmatoarele declaratii sint echivalente:
```
LOGICAL, BYTE, INTEGER*1
INTEGER, INTEGER*2
REAL, REAL*4
DOUBLE PRECISION, REAL*8
```

- Este permisa modificarea conventiei implicite de tip printr-o instructiune de forma:
```
IMPLICIT tip(l,l1-l2...)
```
unde l,l1,l2 sint litere; Exemple:
```
IMPLICIT INTEGER (A,W-Z), REAL(B-F,M-Q)
```

- Instructiunile ENCODE si DECODE se pot utiliza pentru conversia datelor in memorie :
```
ENCODE (a,f) lista
DECODE (a,f) lista
```
unde: "a" este numele unui tablou;
"f" este eticheta instructiunii FORMAT;
"lista" este o lista de variabile similara cu lista din instructiunile READ/WRITE.

Instructiunea DECODE este similara cu instructiunea READ, dar converteste caracterele ASCII din tabloul "a" in conformitate cu formatul specificat.

Instructiunea ENCODE este similara cu instructiunea WRITE, dar converteste valorile variabilelor din lista in ASCII si depune rezultatul in tabloul "a".

- La instructiunea de atribuire se aplica urmatoarele reguli:
    - Atribuirea de expresii logice la o variabila intreaga se face cu extinderea semnului in octetul superior;
    - Atribuirea de expresii intregi sau logice la o variabila reala sau in dubla precizie se face folosind numarul real echivalent rezultatului logic sau intreg;
    - Atribuirea de expresii reale sau dubla precizie la variabile intregi sau logice se face prin trunchiere;
    - Atribuirea de expresii dubla precizie la variabile reale se face prin rotunjire;
    - Operatorii logici .AND.,.OR.,.XOR. realizeaza operatii logice bit cu bit; operatorul .NOT. realizeaza complementul fata de 1 (inversarea logica);

- Numele unui bloc comun trebuie sa fie diferit de numele subprogramelor. Blocul comun blanc nu are nume, deci:

```
      COMMON X,Y
```

este echivalent cu:

```
      COMMON // X,Y
```

- Este posibil ca in subprograme sa fie mai putine variabile decit in programul principal intr-un bloc comun, dar nu mai multe.

- Lungimea unui bloc comun se poate mari printr-o instructiune EQUIVALENCE ca in exemplul urmator:

```
      REAL R(2,2)
      COMMON X,Y,Z
      EQUIVALENCE (Z,R(3))
```

stabileste urmatoarele echivalente:

```
      X = R(1,1)
      Y = R(1,2)
      Z = R(2,1)
        = R(2,2)
```

A se observa ca in instructiunea EQUIVALENCE se poate scrie R(3) in loc de R(2,1)

- Nu se pot echivala cu EQUIVALENCE doua elemente din acelasi tablou sau doua elemente din blocuri COMMON.

- In instructiunea FORMAT sint permise maximum doua nivele de paranteze, inclusiv paranteza obligatorie la orice FORMAT.

- Instructiuni de intrare/iesire:
    - sint permise instructiuni READ si WRITE fara format pentru transferul datelor fara nici o conversie sau editare:

```
READ (n [,ERR=n1] [,END=n2]) lista
WRITE (n [,ERR=n1] [,END=n2]) lista
```

    - o instructiune WRITE scrie o inregistrare logica, care se poate extinde pe mai multe sectoare disc;
    - o instructiune READ nu trebuie sa aiba o lista cu mai multe date decit capacitatea unei inregistrari, dar este posibila citirea partiala a unei inregistrari;
    - in instructiunile READ/WRITE cu format este permis ca, in locul etichetei instructiunii FORMAT, sa se utilizeze un nume de tablou care sa contina sirul de caractere ce descrie formatul:

```
      INTEGER FRMT(4),TAB(3)
      READ(3,300)FRMT
300   FORMAT(4A2)
      READ(3,FRMT)TAB
```

    - numerele logice ale dispozitivelor periferice folosite in

instructiunile READ si WRITE pot fi:
1 : consola;
2 : imprimanta;
3 : consola;
4 : nealocat;
5 : nealocat;
6,...,10 : fisierele secventiale pe discul A cu numele FORT06.DAT,...FORT10.DAT

Semnificatia acestor numere logice poate fi modificata prin subrutina standard OPEN.

- Fisiere disc:
  - se pot crea si exploata atit fisiere secventiale cit si fisiere in acces direct;
  - daca se scrie intr-un fisier existent, atunci se sterge continutul anterior al fisierului (nu este posibila actualizarea unui fisier);
  - pentru fisierele secventiale prima instructiune READ sau WRITE deschide automat fisierul, iar inchiderea fisierului se face la STOP sau printr-o instructiune ENDFILE n;
  - instructiunea REWIND n inchide fisierul cu numarul logic "n" si apoi il redeschide;
  - subrutina OPEN permite deschiderea explicita a unui fisier pentru acces secvential sau direct, cu atribuirea unui nume si unui numar logic fisierului (numarul logic trebuie sa fie cuprins intre 1 si 10).

  Subrutina OPEN primeste ca argumente :
    - numaru logic asociat fisierului
    - numele si tipul fisierului pe 8 si respectiv pe 3 caractere (eventual completate cu blancuri);
    - numarul unitatii de disc pe care se afla fisierul:
      0 pentru discul implicit
      1 pentru discul A:
      2 pentru discul B: s.a.m.d.

Exemplu:

```
CALL OPEN (6,'PLOT    DAT ',2)
```

Subrutina OPEN nu este necesara daca se creaza sau se exploateaza un fisier secvential cu numar logic intre 6 si 10, sub numele implicit FORxx.DAT, unde "xx"=06, 07, 08,..10.

- instructiunile READ si WRITE pentru acces direct au forma urmatoare:

  READ (u,f,REC=n,ERR=m) lista

  unde:

  u = numar logic asociat fisierului (prin OPEN sau implicit);

  n = numarul articolului citit/scris (un articol are 128 octeti deci coincide cu un sector disc);

  m = eticheta instructiunii la care se sare in caz de eroare la citire sau la scriere.

- Prin scriere in acces direct un fisier se poate extinde.

- Functia BACKSPACE nu este recunoscuta de compilatorul F80.

<u>Conventia de apelare a subprogramelor in FORTRAN</u>

Transmiterea de date intre un program Fortran si un subprogram scris in limbaj de asamblare se poate face in doua moduri:
- prin lista de argumente ;
- prin blocuri comune (instructiunea COMMON).

Pentru legarea de subprograme scrise in limbaj de asamblare cu programe FORTRAN este necesara cunoasterea conventiilor de comunicare cu subprogramele apelate.

Indiferent de tipul parametrilor, se transmit adresele de memorie ale acestor parametri prin registre si, eventual, prin memorie dupa cum urmeaza:

a) Daca numarul de argumente este mai mic sau egal cu 3, atunci adresele argumentelor se transmit prin registre dupa cum urmeaza:

adresa argumentului 1 in reg.HL;
adresa argumentului 2 in reg.DE;
adresa argumentului 3 in reg.BC;

b) Daca numarul de argumente este mai mare ca trei atunci adresele argumentelor se transmit astfel:

adresa argumentului 1 in reg.HL;
adresa argumentului 2 in reg.DE;
adresele argumentelor 3, 4,..., printr-o zona de memorie a carei adresa se afla in reg.BC (adresa din BC este adresa octetului inferior din argumentul 3).

Numarul de argumente nu se transmite explicit la subprogram, deci trebuie cunoscut de subprogram.

Rezultatul subprogramelor de tip functie se transmit prin registre, functie de tipul rezultatului:

intregi : valoare rezultat in registrele HL;
reali : valoare rezultat in $AC;
dubla precizie : valoare rezultat in $DAC.

Variabilele globale $AC si $DAC reprezinta zone de memorie utilizate ca acumulator pentru operatii cu numere reale si, respectiv, numere in precizie dubla.

3. FUNCTII DE BIBLIOTECA

F80 recunoaste urmatoarele functii aritmetice de biblioteca (cuprinse in fisierul FORLIB.REL):

Nume	Definitie	Argument	Functie
ABS	|a|	real	real
IABS	|a|	intreg	intreg
DABS	|a|	dubla precizie	dubla precizie
AINT	[a]	real	real
INT	[a]	intreg	intreg
IDINT	[a]	dubla precizie	intreg
AMOD	a1 mod a2	real	real
MOD	a1 mod a2	intreg	intreg
DMOD	a1 mod a2	dubla precizie	dubla precizie
AMAX0	max(a1,a2,...)	intreg	real
AMAX1	max(a1,a2,...)	real	real
MAX0	max(a1,a2,...)	intreg	intreg
MAX1	max(a1,a2,...)	real	intreg
DMAX1	max(a1,a2,...)	dubla precizie	dubla precizie
AMIN0	min(a1,a2,...)	intreg	real
AMIN1	min(a1,a2,...)	real	real
MIN0	min(a1,a2....)	intreg	intreg
MIN1	min(a1,a2,...)	real	intreg
DMIN1	min(a1,a2,...)	dubla precizie	dubla precizie
FLOAT	intreg => real	intreg	real
IFIX	real => intreg	intreg	real
SIGN	sign(a)	real	real
ISIGN	sign(a)	intreg	intreg
DSIGN	sign(a)	dubla precizie	dubla precizie
DIM	a1-min(a1,a2)	real	real
IDIM	a1-min(a1,a2)	intreg	intreg
SNGL	conversia lui a	dubla precizie	real
DBLE	conversia lui a	real	dubla precizie
EXP	e**a	real	real
DEXP	e**a	dubla precizie	dubla precizie
ALOG	ln(a)	real	real
DLOG	ln(a)	dubla precizie	dublaprecizie
ALOG10	log10(a)	real	real
DLOG10	log10(a)	dubla precizie	dubla precizie
SIN	sin(a)	real	real
DSIN	sin(a)	dubla precizie	dubla precizie

COS	cos(a)	real	real
DCOS	cos(a)	dubla precizie	dubla precizie
TANH	tanh(a)	real	real
SQRT	radacina din a	real	real
DSQRT	radacina din a	dubla precizie	dubla precizie
ATAN	arctan(a)	real	real
DATAN	arctan(a)	dubla precizie	dubla precizie
ATAN2	arctan(a1/a2)	real	real
DATAN2	arctan(a1/a2)	dubla precizie	dubla precizie

Observatie:

Cu a, a1, a2,... s-au notat argumentele functiei. Numarul de argumente al unei functii reiese din definitie.

4. MESAJE DE EROARE LA EXECUTIA PROGRAMELOR FORTRAN

Avertismente

IB (INPUT BUFFER LIMIT EXCEEDED) - depasire capacitate buffer de citire);
TL (TOO MANY LEFT PARANTHESES IN FORMAT) - prea multe paranteze in format;
OB (OUTPUT BUFFER LIMIT EXCEEDED) - depasire capacitate buffer de scriere;
DE (DECIMAL EXPONENT OVERFLOW) - exponent mai mare ca 99;
IS (INTEGER SIZE TOO LARGE) - intreg prea mare;
BE (BINARY EXPONENT OVERFLOW) - exponent mai mare ca 38;
IN (INPUT RECORD TOO LONG) - inregistrare prea mare la citire;
OV (ARITHMETIC OVERFLOW) - depasire aritmetica;
CN (CONVERSION OVERFLOW) - depasire la conversie real- intreg;
SN (ARGUMENT TO SIN TOO LARGE) - arg. prea mare la functia SIN;
A2 (BOTH ARGUMENT OF ATAN2 ARE 0) - ambele argumente ale functiei ATAN2 sint 0;
IO (ILLEGAL I/O OPERATION) - operatie de I/E incorecta;
BI (BUFFER SIZE EXCEEDED DURING BINARY I/O) - depasire buffer la citire/scriere binara;
RC (NEGATIVE REPEAT COUNT IN FORMAT) - factor de repetitie negativ in format.

Erori fatale

ID (ILLEGAL FORMAT DESCRIPTOR) - descriptor de format incorect;
FO (FORMAT FIELD WIDTH IS 0) - lungime de cimp nula in format;
MP (MISSING PERIOD IN FORMAT) - lipsa punct in format;
FW (FORMAT FIELD WIDTH IS TOO SMALL) - lungime de cimp prea mica in format;
IT (I/O TRANSMISSION ERROR) - eroare la transfer de I/E;
ML (MISSING LEFT PARANTHESIS IN FORMAT) - absenta paranteza stinga din format;
DZ (DIVISION BY 0, REAL OR INTEGER) - impartire prin 0 la intregi sau reali;
LG (ILLEGAL ARGUMENT TO LOG FUNCTION) - argument ilegal la functia LOG;
SQ (ILLEGAL ARGUMENT TO SQRT FUNCTION) - argument ilegal la functia SQRT;
DT (DATA TYPE DOES NOT AGREE WITH FORMAT SPECIFICATION) - neconcordanta intre tipul variabilelor si specificatia de format;
EF (EOF ENCOUNTERED ON READ) - sfirsit de fisier la citire.

5. EXEMPLE DE PROGRAME FORTRAN

1) Program de listare fisier disc (acces secvential)

```
      BYTE LINIE(72),NUME(11)
      READ(1,9)NUME
9     FORMAT(11A1)
      CALL OPEN(7,NUME,0)
      READ(7,1)LINIE
1     FORMAT(80A1)
      WRITE(1,2) LINIE
2     FORMAT(1X,80A1)
3     READ(7,1,END=9)LINIE
      WRITE(1,2)LINIE
      GOTO 3
9     STOP
      END
```

2) Program de scriere-citire fisier in acces direct

```
      BYTE OP,REC(80),LR,LQ,LW
      DATA LR/'R'/,LQ/'Q'/,LW/'W'/
      CALL OPEN(6,'TEST    DAT',0)
1     WRITE(3,12)
12    FORMAT(' Operatie (Read/Write/Quit):')
      READ(1,11) OP
11    FORMAT(A1)
      IF (OP.EQ.LQ) STOP
      IF (OP.NE.LW .AND. OP.NE.LR) GOTO 1
      WRITE(3,13)
13    FORMAT(' Numar Inregistrare (1-2 cifre):')
      READ(1,14)N
14    FORMAT(I2)
      IF (OP.EQ.LR) GOTO 2
      WRITE (3,15)
15    FORMAT(' Introduceti date (max 80 car.):'/)
      READ(1,16) REC
16    FORMAT(80A1)
      WRITE(6,REC=N) REC
      GOTO 1
2     READ(6,REC=N,ERR=9) REC
      WRITE(3,17) REC
17    FORMAT(1X,80A1)
      GOTO 1
9     WRITE(3,18)
18    FORMAT(' Inregistrare in afara fisierului'/)
      GOTO 1
      END
```

3) Program Fortran cu subrutina ir limbaj de asamblare; datele se transmit prin bloc comun neetichetat (comun blanc)

```
C COMUNICARE FORTRAN-ASAMBLOR PRIN BLOC COMUN
       LOGICAL I,J,K
       COMMON  I,J,K
```

```
10    READ(1,1) I,J
1     FORMAT(2I3)
      IF(I+J .EQ. 0) STOP
      CALL  BIC
      WRITE(3,2) K
2     FORMAT(11X,'Produsul logic =',I3/)
      GOTO 10
      END
```

```
; Comunicare Asamblor -Fortran prin bloc comun
    COMMON / /
I:  DS  1
J:  DS  1
K:  DS  1
    CSEG
    ENTRY  BIC
BIC: LDA  I  ;primul operand
     MOV  B,A
     LDA  J   ;al doilea operand
     ANA  B   ;produs logic in A
     STA  K   ;pune rezultat in blocul comun
     RET
     END
```

4) Program Fortran cu subrutina in limbaj de asamblare; datele se transmit prin lista de parametrii

```
C COMUNICARE FORTRAN-ASAMBLOR PRIN ARGUMENTE
      INTEGER*1 I,J,K
10    READ(1,1) I,J
1     FORMAT(2I3)
      IF(I.EQ.0 .AND. J.EQ.0) STOP
      CALL BIC(I,J,K)
      WRITE(3,2)I,J,K
2     FORMAT(11X,I3,' and',I3,' =',I3/)
      GOTO 10
      END
```

```
; Comunicare asamblor-Fortran prin argumente
    ENTRY  BIC
BIC: MOV   A,M ;valoarea primului parametru
     XCHG      ;HL=bval.par.2
     ANA   M    ;produs logic cu par.2
     STAX  B   ;rezultat in par.3
     RET
     END
```

Observatie:
Exemplele 3) si 4) sint pur didactice deoarece operatorul .AND. din F80 realizeaza produs logic bit cu bit intre operanzi si deci nu este necesara o subrutina in limbaj de asamblare .

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

EDITORUL DE LEGATURI L80 SI BIBLIOTECARUL LIB80

1986

1. EDITORUL DE LEGATURI L80

Programul L80 este un editor de legaturi pentru module obiect relocabile in format .REL,rezultate din compilare FORTRAN (cu F80), din asamblare cu asamblorul M80, din compilare BASIC (cu BASCOM) sau din compilare COBOL.

Linkeditorul L80 poate lega impreuna mai multe module obiect aflate in fisiere de tip .REL cu module extrase din una sau mai multe biblioteci de subrutine relocabile, producind un singur program executabil care se incarca in memorie si care, la cerere, este scris intr-un fisier de tip .COM.

Bibliotecile prelucrate de L80 trebuie create cu un program bibliotecar special numit LIB80.(Programele M80, L80, F80, LIB80 sint create de firma Microsoft).

Listarea numelor modulelor dintr-o biblioteca se poate face cu bibliotecarul sau chiar cu linkeditorul L80, daca se leaga intre ele toate modulele din biblioteca si se foloseste optiunea /M de afisare a simbolurilor globale.

Linkeditorul L80 cauta implicit intr-o biblioteca cu numele FORLIB.REL (necesara pentru modulele generate de compilatorul FORTRAN), dar se poate cere explicit cautarea in alte biblioteci pentru satisfacerea referintelor externe (optiunea /S).

Nu este prevazuta crearea de programe segmentate cu ajutorul linkeditorului L80.

<u>Sintaxa comenzii de linkeditare:</u>

Prin comanda de lansare a linkeditorului se incarca in memorie programul L80 si, eventual, se transmit numele fisierelor de intrare si optiunile. Este posibil ca numele fisierelor si/sau optiunile sa fie introduse pe rind, dupa incarcarea sa in memorie

Sfirsitul introducerii de module obiect si de optiuni este marcat prin optiunea /E (Exit) sau /G (Go), moment in care link-editorul cauta in biblioteca implicita si/sau in bibliotecile date explicit pentru a rezolva referintele externe ramase si pentru a incheia editarea.

In lipsa optiunii /N, linkeditorul incarca in memorie programul creat si nu creaza un fisier disc cu acest program.

Comanda de linkeditare poate avea una din formele:

```
     L80 frel1/opt,frel2/opt,.../E<CR>
sau:
     L80 frel1/opt,frel2/opt,...,fcom/N/E<CR>
```

cind se doreste obtinerea unui fisier obiect de tip .COM, sau:

```
     L80<CR>
     *frel1/opt<CR>
     *frel2/opt<CR>
     */E<CR>
sau:
     L80<CR>
     *frel1/opt<CR>
     *frelD/opt<CR>
         ...
     *fcom/N
     *E
```

cind se doreste obtinerea unui fisier "fcom" de tip .COM

Optiunile "opt" pot avea una din valorile urmatoare:

/D:<adresa>
Stabilire adresa de incarcare a segmentelor de date si de comun; adresa implicita a segmentului de date este 103H.

/E
/E:nume
Terminare linkeditare, eventual cu indicarea numelui simbolului global ce reprezinta adresa de lansare.

/G
/G:nume
Lansare automata in executie dupa linkeditare, eventual cu indicarea numelui adresei de lansare (simbol public).

/M
Listare inceput si sfirsit seggment de date si de program, precum si numele simbolurilor globale (publice si externe).

/P:adresa
Stabilire adresa de incarcare a segmentului de cod din urmatorul modul obiect prelucrat (nu are efect asupra modulului curent). Adresa implicita a segmentului de cod este 100H.

/R
Relansarea de la inceput a linkeditorului, fara reincarcarea sa in memorie (se foloseste dupa o eroare la numele fisierelor de intrare).

fisier/S
Se cere linkeditorului sa caute in biblioteca indicata prin <fisier>, pentru rezolvarea referintelor externe.

/U
Listare inceput si sfirsit segment de date si de program, precum si numele simbolurilor globale nedefinite.

Observatii:

- Optiunile /D si /P pot lipsi; in acest caz linkeditorul plaseaza segmentul de date inaintea segmentului de cod la fiecare modul si introduce o instructiune de salt (JMP) peste segmentul de date. Daca lipseste segmentul de date, atunci segmentul de program urmeaza imediat instructiunii JMP ;
- Adresa de incarcare implicita este 100H;
- Daca se folosesc directive ORG intr-un program, atunci adresele continute in aceste directive sint tratate ca adrese relative la baza de incarcare implicita (100H) si nu ca adrese absolute;
- Adresa primei locatii libere dupa segmentul de date (sau dupa segmentul de cod daca lipseste segmentul de date) este introdusa de linkeditor in simbolul global $MEMRY, daca a fost definit un asemenea simbol in program.

Exemple de comenzi de linkeditare:

1) L80 TEST,TEST/N/E<CR>

Se face relocarea programului obiect citit din fisierul TEST.REL, se creaza un fisier TEST.COM si se revine in sistem.

2) L80 TEST/G<CR>

Se face relocarea programului obiect citit din fisierul TEST.REL, se incarca programul absolut in memorie si se lanseaza in executie.

3) L80 G1,G2<CR>
*GLIB/S<CR>
*GRAF/N/E<CR>

Se leaga impreuna modulele citite din fisierele G1.REL si G2.REL cu subrutinele apelate din aceste module si extrase din biblioteca GLIB; se creaza fisierul GRAF.COM.

```
4) L80<CR>
   *G1<CR>
   *G2<CR>
   *GLIB/S<CR>
   *GRAF/N<CR>
   */E<CR>
```

Acelasi efect cu secventa din exemplul precedent.

2. BIBLIOTECARUL LIB80

Programul bibliotecar LIB80 realizeaza urmatoarele functii:
- Crearea de biblioteci relocabile pornind dela fisiere rezultate din compilari si asamblari diferite;
- Listarea modulelor dintr-o biblioteca impreuna cu toate numele de simboluri globale;
- Extragerea de module obiect din fisiere sau biblioteci in alte fisiere relocabile.

Apelarea bibliotecarului se fac prin comanda:

LIB80

Dupa incarcarea sa in memorie LIB80 afiseaza un asterisc (*) si asteapta comenzi. Formele unei comenzi LIB80 pot fi :

```
*flib=frel1,frel2,../opt
```
sau:
```
*flib/opt
```
sau:
```
*flib=frel1
*frel2
*frel3
...
*/E
```

Numele fisierelor de intrare se pot da fie in aceeasi linie, separate prin virgule, fie pe linii separate.

Notatii folosite:
"flib" este numele fisierului biblioteca creat (poate lipsi);
"frel" este numele unui fisier de intrare de tip .REL;
"opt" sint optiuni pentru bibliotecar.

Un fisier relocabil poate contine unul sau mai multe module obiect. Un modul obiect corespunde unei unitati de program Fortran (program principal, subrutina sau functie) sau unei secvente in limbaj de asamblare ce contine directive ENTRY si eventual alte directive pentru stabilire nume modul (NAME, TITLE).

Bibliotecarul LIB80 concateneaza module obiect si nu fisiere relocabile.

Pentru un fisier relocabil care contine mai multe module obiect se poate preciza numele modulului sau modulelor care se extrag din fisierul respectiv. Daca nu se indica in mod explicit care module se extrag atunci se considera implicit toate modulele din fisierul respectiv.

Specificarea unui singur modul "MOD" din fisierul "FILE":

FILE<MOD> sau FILE<MOD+n> sau FILE<MOD-n>

modulul MOD sau modulul "n" dupa sau inainte de MOD.

Specificarea prin nume a fiecarui modul extras:

FILE<MOD1,MOD2,MOD3>

modulele MOD1, MOD2, MOD3.

Specificarea unei liste de module adiacente:

FILE<MOD1..MOD5>

toate modulele intre MOD1 si MOD5 inclusiv MOD1,MOD5;

FILE<..MOD5>

toate modulele dela inceputul lui FILE pina la MOD5 si MOD5;

FILE<MOD1..>
toate modulele care urmeaza lui MOD1 in FILE, inclusiv MOD1.

Daca nu se specifica explicit fisierul de iesire atunci se considera implicit ca fisier de iesire biblioteca FORLIB.REL.

Optiuni LIB80:

- /L Listarea modulelor dintr-un fisier sau dintr-o biblioteca cu punctele de intrare si referintele externe din fiecare modul.
- /E Iesire din LIB80 in CP/M. Biblioteca creata primeste tipul .REL deoarece pe parcursul crearii are tipul .LIB.
- /R Modifica tipul bibliotecii create din .LIB in .REL; nu se iese.
- /U Listare referinte nerezolvate (referinte inapoi) dupa parcurgerea fisierului specificat.
- /C Sterge biblioteca in curs de creare; reluare LIB80 de la inceput.

Exemple de utilizare LIB80:

1) Crearea unei noi biblioteci GLIB.REL pe baza modulelor din fisierele MOVE.REL, DRAW.REL (provenite din Fortran) si DOT.REL (provenit din asamblare). Subrutina DOT este apelata din MOVE si din DRAW; DRAW apeleaza pe MOVE.

```
A>LIB80
*GLIB=DRAW,MOVE,DOT
*/E
```

2) Adaugarea modulelor din fisierul DASH.REL la biblioteca GLIB:

```
A>LIB80
*TEMP=GLIB,DASH
*TEMP/R
*GLIB=TEMP/E
```

3) Verificarea ordinii corecte a modulelor in biblioteca GLIB:

```
A>LIB80
*GLIB/U
*GLIB/L/E
```

4) Extragerea modulelor MOVA si DRAWA din biblioteca GLIB in fisierul GRAF.REL:

```
A>LIB80
*GRAF=GLIB<MOVEA,DRAWA>/E
```

Observatii:

- Extragerea unor module obiect dintr-o biblioteca sau dintr-un fisier nu sterge aceste module din fisier; nu se poate face o stergere selectiva de module dintr-un fisier relocabil.
- Daca exista anterior pe disc un fisier cu numele bibliotecii create atunci se sterge acest fisier fara nici o avertizare!
- Adaugarea de noi module se face intotdeauna la sfirsitul fisierului destinatie; nu se pot intercala noi module intre alte module dintr-o biblioteca.
- Nu se poate da o comanda de forma:
  GLIB=GLIB,DASH
  cu intentia de adaugare a fisierului DASH.REL la GLIB.REL.
- Nu se recomanda modificarea bibliotecii standard FORLIB.REL.

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

LIMBAJUL SI COMPILATORUL PASCAL/MT+

1986

CUPRINS

1. GENERALITATI 2

2. INSTRUCTIUNI DE UTILIZARE PASCAL/MT+ 4
2.1 Optiuni de compilare in linia de comanda 5
2.2 Optiuni de compilare in programe PASCAAL/MT+ 6

3. UTILIZAREA LINKEDITORULUI LINKMT 8
3.1 Fisiere de comenzi pentru linkeditor 8
3.2 Relatia dintre LINKMT si L80 9

4. COMPILARI SEPARATE (MODULARE) 10

5. SEGMENTAREA PROGRAMELOR PASCAL 11

6. INLANTUIREA DE PROGRAME PASCAL ("CHAINING") 14

7. UTILIZAREA DE SUBRUTINE IN LIMBAJ DE ASAMBLARE 15

8. UTILIZAREA DEPANATORULUI SIMBOLIC 17

9. UTILIZAREA BIBLIOTECARULUI LINKMT 19

10. UTILIZAREA DEZASAMBLORULUI PASCAL/MT+ 20

11. LIMBAJUL PASCAL/MT+. PARTICULARITATI SI EXTINDERI 21
11.1 Extensii PASCAL/MT+ fata de standard 21
11.2 Proceduri si functii predefinite in PASCAL/MT+ 24
11.3 Exemple de programe PASCAL cu fisiere. 28

1. GENERALITATI

Documentatia prezinta modul de utilizare al compilatorului PASCAL si al linkeditorului asociat, produse ale firmei MT MicroSYSTEMS (vers.PASCAL-08 din 1983). Sint prezentate de asemenea si particularitatile limbajului PASCAL/MT+ in raport cu limbajul PASCAL standard.

Fisierele necesare pentru utilizarea limbajului PASCAL/MT+ sint urmatoarele:

```
MTPLUS.COM    - Prima parte a compilatorului
MTPLUS.000    - Segmente ale compilatorului
MTPLUS.001    -          -"-
MTPLUS.002    -          -"-
MTPLUS.003    -          -"-
MTPLUS.004    -          -"-
MTPLUS.005    -          -"-
LINKMT.COM    - Editor de legaturi
MTERRS.TXT    - Lista mesajelor de eroare la compilare
PASLIB.ERL    - Biblioteca relocabila Pascal/MT+
DEBUGGER.ERL  - Depanator simbolic
```

Fisierele urmatoare sint utile dar nu sint absolut necesare:

```
FPREALS.ERL  - Biblioteci relocabile
TRANCEND.ERL -          -"-
RANDOMIO.ERL -          -"-
FULLHEAP.ERL -          -"-
LIBMT.COM    - Bibliotecar pentru format relocabil ERL
DIS8080.COM  - Dezasamblor pentru codul generat de compilator
```

Limbajul PASCAL/MT+ respecta standardul ISO pentru acest limbaj cu urmatoarele extensii fata de standard:

- Noi tipuri de date predefinite: BYTE, WORD, STRING;
- Acces la porturile de I/E prin INP, OUT;
- Adrese absolute de memorie ;
- Operatori logici bit cu bit asupra intregilor;
- Alternativa ELSE la instructiunea CASE;
- Proceduri externe ;
- Proceduri de tratare intreruperi;
- Fisiere disc secventiale si cu acces direct;
- Este posibila compilarea separata pe module;
- Este posibila includerea de cod simbolic 8080 in Pascal/MT+;
- Este posibila legarea de module Pascal cu module in limbaj de asamblare;
- Este posibila segmentarea programelor mari;
- Este posibila inlantuirea de programe la executie;
- Este posibila depanarea simbolica a programelor Pascal;
- Programele generate pot fi executate intr-o memorie ROM;
- Exista un numar mare de proceduri predefinite pentru:
  - operatii la nivel de bit si de octet;
  - operatii asupra sirurilor de caractere;
  - gestiunea dinamica a memoriei;
  - acces la parametrii din linia de comanda;
  - operatii diverse cu fisiere disc la nivel de octet, de inregistrarare sau de bloc in acces secvential si direct.
- Sint prevazute multe optiuni de compilare, printre care:

- recursivitate controlabila pe portiuni;
- generare de cod optimizat pentru Z80;
- reducerea dimensiunii codului generat pe mai multe cai;
- verificari optionale la executie;
- includere de fisiere la compilare,s.a.

Dintre facilitatile limbajului Pascal standard incluse in Pascal MT+ mentionam:

- Date de tip real in binar virgula mobila sau binar-zecimal;
- Date de tip multime ("Set") cu 256 componente;
- Inregistrari cu variante;
- Fisiere TEXT si fisiere de orice tip;
- Utilizarea de functii si proceduri ca parametrii;
- Salt in afara unei proceduri (nerecursive);
- Parametrii la instructiunea PROGRAM;
- Proceduri standard de lucru cu fisiere, etc.

2. INSTRUCTIUNI DE OPERARE PASCAL/MT+

In forma sa cea mai simpla, operarea sistemului PASCAL/MT+ necesita urmatoarele etape:

1.Compilarea programului din fisierul sursa TEST.PAS:

```
PASCAL  TEST
```

Rezultatul compilarii este un fisier relocabil TEST.ERL

2.Linkeditarea programului din fisierul relocabil cu biblioteca PASCAL:

```
LINKMT  TEST,PASLIB/S
```

3.Executaea programului din fisierul TEST.COM:

```
TEST
```

Observatii:

- S-a presupus ca toate fisierele se gasesc pe un acelasi disc, aflat pe unitatea implicita. Este posibil sa se utilizeze un disc pentru compilari si alt disc pentru linkeditare. Numele fisierelor pot fi precedate de numele discului suport;
- Compilatorul considera pentru fisierele sursa tipul implicit .SRC sau tipul implicit .PAS; pot fi folosite si alte tipuri;
- Linkeditorul considera pentru fisierele de intrare tipul implicit .ERL ("Extended Relocatable") si nu admite un alt tip;
- In comanda LINKMT pot apare mai multe fisiere de intrare si/sau mai multe biblioteci, dar biblioteca PASLIB trebuie sa fie ultima in linia de comanda.
- Numele fiserului de tip .COM este acelasi cu numele primului fisier .de tip ERL (programul principal), dar poate fi specificat explicit ca in comanda urmatoare:

```
LINKMT TEST=TEST,PASLIB/S
```

- Optiunea /S utilizata cu numele unei biblioteci semnifica o extragere selectiva din biblioteca respectiva a modulelor necesare si nu includerea intregii biblioteci in programul creat.

- Celelalte biblioteci de module relocabile din sistemul Pascal/MT+ au urmatoarea utilizare:

FPREALS.ERL - Biblioteca aritmetica pentru numere reale in binar virgula mobila, inclusiv intrari-iesiri pentru reali;

TRANCEND.ERL - Biblioteca de functii transcendentale :functii trigonometrice, exponentiala etc.; foloseste rutine din biblioteca FPREALS si de aceea trebuie sa o preceada in linia de comanda pentru linkeditare (LINKMT parcurge o singura data fisierele de intrare, in ordinea intilnirii lor in comanda);

RANDOMIO.ERL - Biblioteca de proceduri pentru acces direct la fisiere disc (acces "aleator");

FULLHEAP.ERL - Biblioteca cu procedurile standard NEW, DISPOSE de gestiune dinamica a memoriei. Biblioteca PASLIB.ERL contine alte doua proceduri NEW si DISPOSE pentru o varianta mai simpla de alocare a memoriei dinamice, fara reutilizarea ei.

ROVLMGR.ERL - O alta varianta a modulului de gestiune a segmentarii ("Overlay Manager") decit cea din PASLIB.

2.1 Optiuni de compilare in linia de comanda

Compilatorul Pascal/MT+ accepta in linia de comanda, dupa numele fisierului de intrare un sir de parametrii de compilare care incepe cu caracterul "$" si continua pina la sfirsitul liniei. Se pot utiliza sau nu spatii intre parametrii succesivi.

Un parametru consta dintr-o litera sau o litera urmata de un alt caracter ce codifica un dispozitiv CP/M. Codificarea folosita este urmatoarea:

X = consola sistem (CON:)
P = imprimanta (LST:)
@ = unitatea de discuri implicita
A,B,.. = unitatea de discuri 0,1,..

Lista alfabetica a optiunilor de compilare:

A Apelare automata a linkeditorului dupa compilare (este necesar un fisier de comenzi .CMD cu numele fisierului sursa). Implicit se revine in sistem dupa compilare

B Numerele reale se reprezinta in BCD (binar-zecimal). Implicit numerele reale se reprezinta in binar virgula mobila.

C Continua compilarea in caz de eroare sintactica. Implicit compilatorul se opreste la fiecare eroare si intreaba daca se continua sau nu compilarea.

D Genereaza informatii pentru depanator in codul obiect si scrie un fisier de tip .PSY pe acelasi disc cu fisierul .ERL. Implicit optiunea este inactiva (nu se genereaza fisier PSY)

Ed Fisierul MTERRS.TXT se afla pe discul "d" (d=@,A,B,..). Implicit fisierul este cautat pe discul curent.

Od Fisierele segmente MTPLUS.00n se afla pe discul "d". Implicit toate segmentele pe discul curent.

Pd Scrie fisierul de tip .PRN pe dispozitivul "d" (d=X,P,@,A,B). Implicit nu se genereaza fisier .PRN (lista de compilare)

Q Nu se afiseaza mesaje la consola pe parcursul compilarii. Implicit compilatorul traseaza desfasurarea compilarii.

Rd Scrie fisierul de tip .ERL pe discul "d". Implicit fisierul .ERL se scrie pe discul cu fisierul sursa.

Td Scrie fisierul de manevra PASTEMP.TOK pe discul "d". Implicit fisier de lucru pe discul curent

V Afisare nume de proceduri si functii pe masura compilarii. Implicit nu se afiseaza numele procedurilor in faza 1.

X Include in fisierul .ERL informatii pentru dezasamblor. Implicit nu se genereaza aceste date in formatul .ERL

Z Genereaza cod optimizat pentru procesorul Z80. Implicit genereaza numai instructiuni I8080.

@ Interpreteaza caracterul "@" ca sinonim pentru "^". Implicit caracterul "@"

este tratat ca litera si difera de ^

Exemple de utilizare optiuni de compilare:

```
PASCAL TEST $QCZ
PASCAL TEST $RB PX D
```

Compilatorul Pascal/MT+ afiseaza in fiecare faza urmatoarele:

In faza 0 (de analiza sintactica):
- Cite un caracter "+" la fiecare 16 linii sursa si numarul total de linii sursa;

In faza 1 (construirea tabelei de simboli):
- Memoria disponibila la inceputul acestei faze;
- Marimea tabelei de simboli utilizator;
- Cite un caracter "#" la fiecare functie sau procedura
- Memoria disponibila la sfirsitul fazei 1;

In faza 2 (generarea de cod obiect):
- Numele fiecarei proceduri si lungimea codului generat;
- Coduri si (daca exista fisier MTERRS.TXT) mesaje de eroare;
- Numar de linii sursa compilate;
- Numar de erori detectate
- Numar de octeti de cod (in zecimal);
- Numar de octeti de date (in zecimal).

2.2 Optiuni de compilare in programe Pascal/MT+

Aceste optiuni se introduc sub forma de comentarii speciale care incep cu caracterul "$":

{ $p }   sau   (* $p *)

unde "p" este numele parametrului (o litera), urmat eventual de "+" (optiune activa) sau "-" (optiune inactiva) sau de o valoare specifica fiecarei optiuni de compilare.
Intr-un comentariu se poate introduce o singura optiune de compilare.
Lista alfabetica a optiunilor de compilare - comentariu:

Cn — Generare de instructiuni RST n pentru apelare subrutine de lucru cu numere reale. Implicit se genereaza instructiuni CALL ptr aceste rutine.

E+/E- — Genereaza sau nu atributul de "punct de intrare" pentru toate variabilele globale si numele de proceduri/functii. Implicit E+ ,deci toate sint simboluri externe.

Inume — Include la compilare fisierul sursa cu numele indicat, in locul unde se afla optiunea.

Kn — Elimina o serie de nume de proceduri standard din tabela de simboluri predefinite in functie de valoarea lui n (0..15). Implicit tabela de simboli contine toate numele de functii si de proceduri predefinite in Pascal MT+.

L+/L- — Activeaza/dezactiveaza listarea programului sursa in faza 1 (se poate utiliza de mai multe ori ptr. listarea selectiva). Implicit L+, deci se listeaza tot programul sursa.

P Insereaza caractere FF (FormFeed=0CH) in fisierul .PRN

R+/R- Genereaza sau nu cod suplimentar pentru verificarea la executie a depasirii indicilor de tablou si a subdomeniului pentru variabile de tip m..n. Implicit R- deci nu se fac aceste verificari la executie.

Qn Generare de instructiuni RST n pentru incarcari si memorari in proceduri recursive. Implicit se genereaza instructiuni CALL.

S+/S- Alocare de memorie in stiva (S+) sau statica (S-) pentru parametrii si ptr. variabilele locale din proceduri/functii. Procedurile recursive trebuie compilate cu S+.
Implicita este optiunea S- deci proceduri nerecursive!

T+/T- Verifca sau nu corespondenta stricta a tipurilor de date la executie, deci respectarea standardului. Implicita este optiunea T- deci varianta mai toleranta.

X+/X- Verifica sau nu la executie exceptiile aritmetice:impartire prin zero la intregi si la reali,depasire superioara sau inferioara la operatii cu reali, depasire lungime maxima la sirurile de caractere (tipul STRING). Implicta este optiunea T+ deci se fac verificarile.

W+/W- Verificare portabilitate (respectare standard) cu mesaje de avertisment (Warning) in caz de nerespectare. Se foloseste impreuna cu optiunea T+. Implicita este optiunea W- deci nici o verificare.

Z$nnnn Initializare registru de stiva SP cu adresa hexa "nnnn". Implicit reg. SP este incarcát cu adresa de sfirsit a zonei TPA ,luata dela locatiile 6 si 7.

Observatii:

- Optiunile E,K si S trebuie folosite inaintea cuvintelor cheie PROGRAM sau MODULE (chiar la inceputul textului PASCAL);
- Optiunea Z trebuie pusa in programul principal, inante de BEGIN;
- OptiunileR,X,T,W pot fi utilizate de mai multe ori intr-un acelasi program cu valori "+" si "-" pentru activarea si dezactivarea selectiva pe portiuni a verificailor respective;
- Interpretarea valorilor lui "n" la optiunea Kn este :

0 ROUND, TRUNC, EXP, LN, ARCTAN, SQRT, COS, SIN
1 COPY, INSERT, POS, DELETE, LENGTH, CONCAT
2 GNB, WNB, CLOSEDEL, OPENX, BLOCKREAD, BLOCKWRITE
3 CLOSE, OPEN, PURGE, CHAIN, CREATE
4 WRD, HI, LO, SWAP, ADDR, SIZEOF, INLINE, EXIT, PACK, UNPACK
5 IORESULT, PAGE, NEW, DISPOSE
6 SUCC, PRED, EOF, EOLN
7 TSTBIT, CLRBIT, SETBIT, SHR, SHL
8 RESET, REWRITE, GET, PUT, ASSIGN, MOVELEFT, MOVERIGHT, FILL
9 READ, READLN
10 WRITE, WRITELN
11 nefolosit
12 MAXAVAIL, MEMAVAIL
13 SEEKREAD, SEEKWRITE
14 RIM85, SIM85, WAIT
15 READHEX, WRITEHEX

3. UTILIZAREA LINKEDITORULUI LINKMT

Sintaxa generala a comenzii de linkeditare este urmatoarea:

```
LINKMT [prog=] prog,modul,modul,...,PASLIB/optiuni
```

cu interpretarea urmatoare:

- Se poate da, optional, un nume fisierului de iesire (de tip COM) separat prin "=" de numele fisierelor de intrare. In lipsa unui nume explicit, fisierul de iesire primeste automat numele primului fisier de intrare, care trebuie sa fie programul principal.
- Fisierele de intrare notate cu "modul" pot fi biblioteci de module relocabile sau module individuale rezultate din compilare sau din asamblare cu M80, dar numai de tipul .ERL sau .CMD.
- Sint admise o serie de optiuni de linkeditare, precedate de caracterul "/"; aceste optiuni sint de doua tipuri:
  - optiuni individuale la fiecare fisier: /S,/F,/O
  - optiuni globale, scrise la sfirsitul liniei: /L,/M,/E,/P,/D .

Lista optiunilor permise la linkeditare:

/L	Listare adresa de cod si de date la fiecare modul
/E	Listare nume puncte de intrare, inclusiv din biblioteci
/M	Listare puncte de intrare sub forma tabelara
/P:nnnn	Stabileste adresa "nnnn" pentru sectiunea de cod
/D:nnnn	Stabileste adresa "nnnn" pentru sectiunea de date
/W	Se scrie un fisier de tip .SYM utilizabil de catre SID
/F	Fisierul anterior este un fisier de tip .CMD
/S	Fisierul anterior este o biblioteca cautata selectiv
/O:n	Atribuie numarul "n" unui segment (Overlay)
/Vm:nnnn	Stabileste adresa "nnnn" pentru segmentul "m"
/X:nnnn	Stabileste memoria de date suplimentara la un segment

3.1 Fisiere de comenzi pentru linkeditor

Linkeditorul LINKMT poate citi numele fisierelor prelucrate fie de la consola fie dintr-un fisier de comenzi de tip .CMD.

Fisierul de comenzi CMD contine o lista de nume de fisiere scrise pe linii separate sau pe o singura linie da separate prin virgule. Numarul maxim de caractere admise este de 256. Fisierul de comenzi CMD trebuie urmat de optiunea /F.

<u>Exemplu:</u>

In locul comenzii:

```
LINKMT  CALC,TRANCEND,FPREALS,PASLIB/S
```

se poate introduce dela consola comanda urmatoare:

```
LINKMT  CALC/F
```

daca a fost creat in prealabil urmatorul fisier CALC.CMD :

```
CALC
TRANCEND,FPREALS
PASLIB/S
```

Aceasta solutie este preferabila ca timp de raspuns fata de utilizarea unui fisier de comenzi de tip .SUB si a programului SUBMIT din CP/M.

3.2 Relatia dintre LINKMT si L80

Formatul relocabil .ERL acceptat de LINKMT nu este identic dar este compatibil cu formatul relocabil .REL acceptat de L80.
Formatul ERL contine majoritatea inregistrarilor din formatul REL cu exceptia a citeva inregistrari, corespunzatoare blocurilor de comun (COMMON), initializarii de date in sectiunea de date si cererilor de cautare in biblioteca. Spre deosebire de formatul REL ,in formatul ERL se impune o anumita ordine a inregistrarilor si se accepta nume publice de 7 caractere (fata de 6 la REL).
LINKMT cere ca dimensiunea sectiunilor de program si de date sa preceada primul octet de date care se incarca; aceasta conditie este indeplinita de catre modulele relocabile generate de M80 dar nu si de catre modulele generate de compilatorul Fortran F80.
Consecintele practice ale acestpor diferente sint urmatoarele:

- Linkeditorul L80 nu accepta formatul .ERL, dar este posibil sa se transforme module ERL generate de Pascal/MT+ in module REL acceptate de L80, folosind bibliotecarul LIBMT;
- Nu se pot lega impreuna module Pascal cu module Fortran;
- Modulele obiect generate de asamblorul M80 pot fi legate cu module Pascal folosind linkeditorul LINKMT, daca sint respectate urmatoarele conditii:
- modulele generate de M80 au extensia .ERL si nu .REL;
- modulele M80 nu folosesc directivele COMMON,.REQUEST si nici directive DB, DW dupa DSEG.

Avantajele utilizarii linkeditorului LINKMT fata de L80 sint:

- Se pot lega programe mai mari decit cu L80;
- Se pot genera programe segmentate;
- Se poate genera si format .HEX, pe linga formatul .COM;
- Fisierul .COM este in general mai scurt deoarece nu se scot pe disc zone de memorie neinitializate .

4. COMPILARI SEPARATE (MODULARE)

Unitatea de compilare in sistemul Pascal/MT+ este fie programul fie modulul Pascal. Un modul are o structura asemanatoare unui program:

- incepe cu o instructiune speciala: MODULE nume;
- poate avea o parte de declaratii pentru etichete, constante, tipuri de date si variabile;
- contine declaratii de proceduri si/sau de functii;
- se termina prin cuvintul cheie MODEND urmat de un punct(".").

Spre deosebire de un program standard Pascal nu este necesara intr-un modul o secventa de instructiuni intre BEGIN si END.

Variabilele declarate la inceputul unui modul pot fi tratate ca variabile locale sau ca variabile globale (externe) in functie de optiunea de compilare $E:

{$E-} inseamna ca variabilele sint considerate locale

{$E+} inseamna ca variabilele sint considerate globale

Prin aceste variabile se poate comunica intre module separate.

Pentru utilizarea de variabile sau functii compilate in alte module decit cel care le apeleaza, se utilizeaza atributul extern (EXTERNAL) in declaratiile de variabile sau de proceduri:

```
 1) VAR I,J,K : EXTERNAL INTEGER;
        CX : EXTERNAL RECORD RE,IM:REAL END;
 2) EXTERNAL PROCEDURE SORT(VAR Q:LIST; LEN:INTEGER);
 3) EXTERNAL FUNCTION IOTEST:INTEGER;
```

Declaratiile de variabile sau proceduri externe pot apare numai la nivelul cel mai exterior dintr-un program sau dintr-un modul.

Concordanta declaratiilor externe din module diferite referitoare la aceleasi vaiabile, functii/proceduri si parametrii de proceduri cade in sarcina programatorului,deoarece nu poate fi verificata nici de compilator , nici de linkeditor.

Procedurile scrise in limbaj de asamblare si apelate din Pascal trebuie de asemenea declarate ca proceduri externe.

Numele de simboluri globale Pascal/MT+ sint limitate la 7 caractere semnificative sau 6 caractere pentru limbaj de asamblare.

5. SEGMENTAREA PROGRAMELOR PASCAL

Un program mare, care nu incape in intregime in memorie poate fi segmentat intr-un segment radacina, rezident in ,memorie, si mai multe segmente de suprapunere ("Overlays"), incarcate succesiv la aceleasi adrese de memorie.

Incarcarea si reincarcarea segmentelor nerezidente nu trebuie programata exlicit de catre utilizator deoarece este prevazut in biblioteca PASLIB un modul special de gestiune a segmentarii.
Acest modul verifica, la apelarea unei proceduri dintr-un segment nerezident, daca segmentul respectiv este in memorie, iar in caz contrar asigura incarcarea de pe disc a acestui segment.

Programatorul are sarcina sa defineasca continutul fiecarui segment, numarul segmentului, adresa de incarcare in memorie si alte informatii necesare pentru gestiunea automata a segmentelor.

Un program segmentat ocupa in memorie mai multe zone adiacente:

- zona segmentului radacina ("Root area"), la adrese mici;
- zona sau zonele de suprapunere ("Overlay area"), unde se incarca segmentele paralele nerezidente;
- zona de date a segmentului radacina (precizata prin /D);
- zona de date a segmentelor nerezidente;
- zona de alocare dinamica, utilizata prin NEW si DISPOSE.

In cazul cel mai simplu exista o singura zona de incarcare a segmentelor nerezidente, dimensionata in functie de segmentul cel mai mare dintre segmentele paralele aduse in zona respectiva. Pot exista 16 zone de suprapunere diferite, succesive.

Numarul de segmente paralele asociate cu o zona de suprapunere este de cel mult 16 segmente.

Un segment poate contine unul sau mai multe module compilate separat de modulele din alte segmente si constituie un fisier.

Incarcarea unui segment nerezident se face in urma apelarii unei proceduri continute in segmentul respectiv, apel care poate proveni din radacina sau dintr-un alt segment, inclusiv din segmente paralele.

La declararea procedurilor externe din alte segmente trebuie precizat si numarul segmentului conform sintaxei:

```
EXTERNAL n PROCEDURE ...  sau EXTERNAL n FUNCTION ...
```

unde "n" este numarul segmentului ce contine procedura/functia.

Segmentele unui program se linkediteaza separat si trebuie sa aiba toate acelasi nume de fisier dar cu extensii diferite: segmentul radacina are extensia necesara numerotarea in continuare a segmentelor, dar segmentele 1-16 sint incarcate in zona de suprapunere 1, segmentele 17-32 in zona 2, s.a.m.d.

Inainte de linkeditarea finala a segmentelor este necesara o linkeditare pentru determinarea lungimii sectiunilor de cod si de date ale fiecarui segment.

Comanda de linkeditare a radacinii are forma urmatoare:

```
LINKMT prog,<alte module>,PASLIB/S/Vn:mmmm/D:dddd/X:pppp
```

unde:
"prog" este numele fisierului radacina (si numele segmentului)
"n"   este numarul zonei de suprapunere (in hexazecimal)
"mmmm" este adresa de memorie a zonei n (in hexa)
"dddd" este adresa sectiunii de date (in hexa) si poate lipsi
 pppp" este dimensiunea memoriei necesare pentru datele locale

din segmentele nerezidente si poate lipsi.

Comanda de linkeditare a unui segment nerezident are forma:

```
LINKMT prog=prog/O:n,<alte module>,PASLIB/S/P:mmmm
```

Este necesar ca linkeditarea radacinii sa se faca inainte de celelalte segmente, deoarece optiunea /V are drept efect generarea unui fisier "prog.SYM" care contine numele simbolice globale, fisier care este necesar la linkeditarea celorlalte segmente: optiunea /O:n se aplica asupra fisierului de tip .SYM si nu asupra unui fisier de tip .ERL.

Adresa zonei sau zonelor de suprapunere se poate determina printr-o linkeditare prealabila a fisierului radacina, fara optiunea /V, si notarea lungimilor sectiunii de cod si de date.
Daca nu s-a folosit optiunea /D atunci adresa primei zone de suprapunere este egala sau mai mare cu 100H + cod + date (100H este adresa de incarcare a intregului program).Daca s-a folosit optiunea /D pentru a plasa sectiunea de date la alta adresa, atunci adresa zonei de suprapunere este cel putin egala cu 100H + cod , unde "cod" este lungimea sectiunii de cod a radacinii.

Adresa astfel calculata apare atit la optiunea /V pentru radacina cit si la optiunea /P de la fiecare segment suprapus.

Optiunea /X este necesara numai atunci cind programul segmentat foloseste procedurile standard de gestiune dinamica a memoriei NEW , DISPOSE. Ea specifica memoria necesara pentru datele locale din segmentele suprapuse si permite linkeditorului sa determine adresa de inceput a memoriei dinamice . Dimensiunea acestei zone este egala cu suma zonelor de date din segmentele incarcate la aceeasi adresa.

Este posibil ca unele segmente, sau toate, sa se afle pe alte unitati de disc decit unitatea de unde s-a incarcat radacina; in acest caz se va folosi procedura @OVS pentru precizarea unitatii pe care se afla un segment. Procedura @OVS se va declara astfel:

```
EXTERNAL PROCEDURE @OVS(n:INTEGER;d:CHAR);
```

unde: "n" este numarul segmentului
"d" este numele unitatii de disc (@,A,B,..)

In mod normal nu este permisa apelarea unui segment S2 dintr-un alt segment paralel S1 deoarece nu este asigurata reincarcarea automata a lui S1 dupa terminarea lui S2 si deci nu se face revenirea corecta din S2 in S1 (aceasta reincarcare cosuma timp si ar fi inutila in majoritatea cazurilor). Pentru a fi posibila apelarea unui segment din alt segment paralel trebuie legata la programul principal o alta varianta a modulului de gestiune a segmentarii, aflata in fisierul ROVLMGR.ERL, in locul celui din PASLIB.

Linkeditarea unui program segmentat trebuie refacuta numai daca s-au efectuat modificari in radacina; daca s-a modificat un segment atunci este suficienta linkeditarea segmentului respectiv

<u>Exemplu de program segmentat:</u>

```
{Fisierul PROG.PAS}
PROGRAM MAINPROG;
VAR I:INTEGER; CH:CHAR;
EXTERNAL [1] PROCEDURE OVL1;
EXTERNAL [2] PROCEDURE OVL2;
BEGIN
 REPEAT
  WRITE('Introduceti o litera: A/B/Q');
  READ(CH); CASE CH OF
   'A','a' : BEGIN I:=1; OVL1 END;
```

```
   'B','b' : BEEGIN I:=2; OVL2 END
   ELSE; END {CASE}
 UNTIL CH IN ['Q','q'];
 WRITELN('Stop')
END.


{Fisierul MOD1.PAS}
MODULE OVERLAY1;
VAR I:EXTERNAL INTEGER;
PROCEDURE OVL1;
BEGIN WRITELN('In segmentul 1 I=',I) END;
MODEND.


{Fisierul MOD2.PAS}
MODULE OVERLAY2;
VAR I:EXTERNAL INTEGER;
PROCEDURE OVL2;
BEGIN WRITELN('In segmentul 2 I=',I) END;
MODEND.
```

Comenzile de linkeditare pentru acest program sint:

```
LINKMT PROG,PASLIB/S/D:8000/V1:4000/X:40
LINKMT PROG=PROG/O:1,MOD1,PASLIB/S/P:4000/L
LINKMT PROG=PROG/O:2,MOD2,PASLIB/S/P:4000/L
```

Observatii:

- Valoarea 4000H pentru adresa zonei de suprapunere 1 este arbitrara si nu are legatura cu lungimea radacinii;
- Valoarea 8000H pentru sectiunea de date a radacinii este tot arbitrara;
- Optiunea /L nu este necesara, da permite punerea in evidenta a faptului ca la segmentele nerezidente nu se adauga acele module din PASLIB care au fost deja introduse in radacina;
- Fisierele create se numesc PROG.COM, PROG.001, PROG.002;
- Executind acest program prin comanda PROG se poate observa ca, introducind succesiv aceeasi litera nu se reincarca segmentul respectiv de fiecare data, dar introducind secventa A,B,A,B,..se reincarca alternativ cele doua segmente.

6. INLANTUIREA DE PROGRAME PASCAL ("CHAINING")

Inlantuirea de programe poate constitui o alternativa la segmentare pentru programe mari.

Un program poate incarca de pe disc un alt program si sa-i predea controlul dupa cum urmeaza:

- Declara o variabila fisier fara tip (FILE;) si asociaza aceasta variabila cu numele extern al fisierului ce contine un at program folosind procedura ASIGN;
- Initializeaza (deschide) fisierul program cu procedura RESET;
- Incarca si lanseaza programul incarcat folosind procedura CHAIN cu numele variabilei fisier ca argument.

Transmiterea de date intre doua programe inlantuite se face fie:

- prin variabile globale, cu acelasi nume si plasate la aceleasi adrese (cu optiunea /D la linkeditare);
- prin variabile absolute de memorie, cu adrese identice in cele doua programe.

In ambele cazuri, programatorul trebuie sa se asigure ca programul inlantuit nu se incarca peste datele comune, mai ales atunci cind un program scurt se inlantuie cu un program lung.

Pentru conservarea memoriei dinamice la o inlantuire, trebuie ca variabila SYSMEM (care indica inceputul memoriei dinamice) sa fie declarata ca EXTERNAL INTEGER.

<u>Exemplu de programe inlantuite:</u>

```
{Primul program - fisier CHAIN1.COM}
PROGRAM CHAIN1;
TYPE COMM=RECORD I,J:INTEGER END;
VAR GLOBALS:ABSOLUTE [$8000] COMM; CHFILE : FILE;
BEGIN
 WITH GLOBALS DO
  BEGIN I:=1; J:=-1 END;
 ASSIGN (CHFILE,'A:CHAIN2.COM'); RESET (CHFILE);
 IF IORESULT=255 THEN BEGIN
   WRITELN('Nu exista fisierul CHAIN2.COM'); EXIT END;
 CHAIN (CHFILE)
END.
```

```
{Al doilea program - fisier CHAIN2.COM}
PROGRAM CHAIN2;
TYPE COMM=RECORD I,J:INTEGER END;
VAR GLOBALS: ABSOLUTE [$8000] COMM;
BEGIN
 WITH GLOBALS DO
  WRITELN ('I=',I,'J=',J);
END.
```

7. UTILIZAREA DE SUBRUTINE IN LIMBAJ DE ASAMBLARE

Editorul de legaturi LINKMT poate lega module obiect provenite din compilare PASCAL (MT+) cu module obiect rezultate din asamblare cu M80, daca sint indeplinite urmatoarele conditii:
- Simbolurile globale nu depasesc 6 caractere in lungime;
- Nu se utilizeaza caracterul "$" in nume simbolice globale;
- Nu se utilizeaza directiva COMMON in lb. de asamblare;
- Nu se utilizeaza directive DB,DW in sectiunea de date (DSEG).

Comunicarea de date intre Pascal si asamblor se poate face:
- prin variabile globale Pascal, sau
- prin parametrii de proceduri sau functii.

Variabilele globale Pascal (declarate in programul principal) pot fi utilizate din limbaj de asamblare daca sint declarate ca simboluri externe (cu directiva EXTRN sau PUBLIC) si daca s-a utilizat optiunea {$E+} la compilarea Pascal.

Pentru a utiliza corect continutul variabilelor Pascal trebuie sa se cunoasca modul de alocare a adreselor de memorie pentru fiecare tip de variabila Pascal.

Variabilele globale Pascal sint plasate in memorie la adrese succesive, in ordinea declararii lor, astfel:
- cite un octet pentru variabile de tip CHAR, BYTE, BOOLEAN;
- cite doi octeti ptr. variabile de tip INTEGER,WORD : intii octetul inferior si apoi octetul superior;
- cite 4 octeti ptr variabile de tip REAL (binar virgula mobila);
- cite 32 de octeti ptr variabile de tip SET;
- intre 1 si 256 octeti pentru variabile de tip STRING : primul octet contine lungimea efectiva a sirului.

Componentele unui tablou cu doua sau mai multe dimensiuni sint memorate pe linii: linia 1, linia 2...etc.

Parametrii procedurilor si functiilor se transmit prin stiva:
- Adresa de revenire din procedura in virful stivei, pe 2 octeti;
- Valoarea sau adresa parametrilor, in ordine inversa :ultimul parametru se afla imediat dupa adresa de revenire.

Pentru fiecare parametru se introduc in stiva unul sau mai multe cuvinte de cite 2 octeti, inclusiv pentru variabile cu lungime de un octet: pentru parametrii scalari care nu sint precedati de VAR se transmit valorile parametrilor; pentru parametrii nescalari si pentru parametrii scalari precedati de VAR se pun in stiva adresele parametrilor.

<u>Exemplu:</u>

```
PROCEDURE ASM(I,J:INTEGER; VAR Q:STRING; C,D:CHAR);
```

La intrarea in procedura ASM stiva are urmatorul continut:

```
(SP)   : adresa de revenire
(SP+2) : D,0
(SP+4) : C,0
(SP+6) : adresa lui Q
(SP+8) : J (octet inferior,octet superior)
(SP+10): I (octet inferior, octet superior)
```

Subrutina in limbaj de asamblare trebuie sa scoata din stiva toti parametrii inainte de revenirea in programul apelant.

Rezultatul unei functii (ca valoare) ramine in virful stivei dupa revenirea

din subprogramul functie:
- se scoate adresa de revenire din stiva si se pastreaza;
- se scot toti parametrii din stiva;
- se pune rezultatul functiei in stiva;
- se pune inapoi in stiva adresa de revenire.

In limbaj de asamblare sint permise numai functii cu rezultat scalar de tip INTEGER, REAL, BOOLEAN, CHAR.

<u>Exemple de proceduri si functii in limbaj de asamblare:</u>

PEEK:citeste din memorie un octet; POKE:scrie un octet in memorie

```
{fisier de tip .PAS}
PROGRAM PEEKPOKE;
TYPE BYTEPTR=^BYTE;
VAR ADDRESS:INTEGER; OPER:INTEGER;
    BBB:BYTE; PPP:BYTEPTR;
EXTERNAL PROCEDURE POKE (B:BYTE;P:BYTEPTR);
EXTERNAL FUNCTION PEEK (P:BYTEPTR):BYTE;
BEGIN REPEAT
 WRITE  ('Introduceti o adresa');
 READLN(ADDRESS); PPP:=ADDRESS;
   WRITE('Operatie:1=PEEK ,2=POKE');
 READLN (OPER);
 IF OPER=1 THEN WRITELN (ADDRESS,'contine',PEEK(PPP))
  ELSE IF OPER=2 THEN BEGIN
   WRITE('Introduceti un octet de date:');
   READLN (BBB); POKE (BBB,PPP) END
 UNTIL FALSE
END.

{fisier de tip .MAC}
     PUBLIC    PEEK,POKE
PEEK:POP  B    ;scoate adresa de revenire in BC
     POP  H    ;valoare parametru in HL (BYTEPTR)
     MOV  E,M  ;continutul adresei de memorie in DE
     MVI  D,0
     PUSH D    ;rezultatul functiei in stiva
     PUSH B    ;adresa de revenire in stiva
     RET

POKE:POP  B    ;scoate adresa de revenire in BC
     POP  H    ;al doilea parametru (BYTEPTR) in HL
     POP  D    ;primul parametru in E (D=0)
     MOV  M,E  ;pune octet in memorie
     PUSH B    ;pune adresa de revenire in stiva
     RET
     END
```

8. UTILIZAREA DEPANATORULUI SIMBOLIC

Depanatorul Pascal/MT+ se prezinta sub forma unui fisier relocabil numit DEBUGGER.ERL, care se leaga impreuna cu programul depanat astfel:

```
LINKMT prog=DEBUGGER,prog,PASLIB/S
```

unde "prog" este numele fisierului ce contine programul depanat, nume care se atribuie si fisierului de tip COM creat.

La executie, controlul este preluat de catre depanator.

Pentru o depanare simbolica este necesar ca la compilare sa se utilizeze optiunea $D in linia de comanda; aceasta optiune are ca efect generarea unui fisier de tip .PSY pe acelasi disc cu fisierul .ERL. Linkeditorul LINKMT prelucreaza fisierul .PSY si produce un fisier de tip .SYP, care contine numele simbolice din programul Pascal cu adresele de memorie asociate. Acest fisier este utilizat de catre depanator, permitind utilizarea de referinte simbolice in comenzile de depanare pentru variabile, proceduri si functii (in locul unor adrese de memorie).

In cazul procedurilor/functiilor recursive (compilate cu $S) nu se pot vizualiza variabilele locale deoarece sint alocate in stiva si nu au alocate adrese permanente.

Comenzile depanatorului au un cod mnemonic de doua litere urmat de un parametru. Parametrul poate fi:
- un nume de variabila;
- un nume de variabila precedat de un nume de procedura, separate prin caracterul ":";
- un numar zecimal sau un numar hexazecimal cu prefixul "$";
- un nume sau un numar urmate de caracterul "^" (semnul de adresare indirecta) indica continutul unei adrese;
- o expresie formata dintr-un nume sau un numar urmat de operatorul "+" sau "-" si de un alt numar.

<u>Exemple:</u>

Fie urmatorul fragment de program Pascal:

```
TYPE PAOC=ARRAY [1..40] OF CHAR;
VAR ABC:INTEGER; PTR:^PAOC;
```

In comenzile de depanare se pot utiliza urmatorii parametrii:

ABC	valoarea variabilei ABC (un numar intreg)
PTR	valoarea variabilei PTR (o adresa)
PTR^	tot tabloul dela adresa continuta in PTR
ABC+10	continutul locatiei cu adresa var. ABC plus 10 octeti
PTR^+10	valoarea caracterului PAOC[11]
ABC-2	continutul locatiei cu adresa var. ABC minus 2
$1A07	continutul adresei absolute hexa 1A07H
$2C50^	32 de octeti dela adresa continuta in locatia 2C50H
$1FFD+$5B	32 de octeti dela adresa 2058H
$2C50^+10	32 de octeti dela adresa continuta in locatia 2C50H adunata cu 10
PROC:I	valoarea variabilei locale I din procedura PROC
PROC:P^+4	continutul locatiei cu adresa calculata din continutul variabilei P plus 4 din procedura PROC.

Comenzile depanatorului Pascal/MT+:

Comenzi de afisare a memoriei:

```
DVnume   Afiseaza valoarea variabilei "nume"
DVnume^  Afiseaza valorile dela adresa continuta in var."nume"
DIpar    Afiseaza un intreg
DCpar    Afiseaza un caracter
DLpar    Afiseaza un boolean (var. logica)
DRpar    Afiseaza un real
DBpar    Afiseaza un byte (un octet)
DWpar    Afiseaza un cuvint  (Word)
DSpar    Afiseaza un sir de caractere (String)
DXpar    Afiseaza o structura pe lungime de 320 octeti
DXpar,n  Afiseaza o structura pe lungime de "n" octeti
SEpar    Afiseaza  si eventual modifica continutul  unor  locatii
         de memorie;  similara cu comanda "S" din DDT, se iese cu
         ".")
```

Comenzi de executie controlata a programului depanat:

```
TR sau T Trasare: se executa o linie de program
Tn        Trasare: se executa "n" linii de program
BE        Executare program dela inceput (Begin Execution)
CO        Continuae executie dupa un punct de oprire
SBnume    Stabilire punct de oprire la inceputul unei proceduri
          (Set Breakpoint)
RBnume    Stergere punct de oprire dela procedura "nume"
          (Remove Breakpoint)
E+        Activeaza  afisarile  la intrarea si  la  iesirea  din
          fiecare procedura sau functie (implicit activ)
E-        Dezactiveaza afisarile la intrari/iesiri din proceduri
PN        Afiseaza numele de proceduri din fisierul .SYP
VNnume    Afiseaza numele variabilelor din procedura "nume"
??        Afiseaza lista comenzilor depanatorului din fisierul
          DBUGHELP.TXT
```

9. UTILIZAREA BIBLIOTECARULUI LINKMT

Bibliotecarul Pascal/MT+ are doua functii:
- Crearea de biblioteci prin concatenarea de fisiere .ERL;
- Trecerea de la formatul .ERL la formatul .REL acceptat DE L80.

Programul bibliotecar este apelat prin comanda:

```
LIBMT fisier
```

unde "fisier" este numele unui fisier de tip .BLD (tipul .BLD nu trebuie specificat explicit in comanda), creat cu un editor.

Fiecare linie din fisierul .BLD contine un singur nume de fisier dupa cum urmeaza:
- Numele fisierului de iesire pe prima linie;
- Numele fisierelor de intrare pe liniile urmatoare.

Daca se doreste crearea unui fisier de iesire compatibil cu L80 atunci este necesar ca pe prima linie sa se scrie "L80", pe linia a doua numele fisierului de iesire, samd.

<u>Exemple de fisiere BLD:</u>

1) Pentru utilizarea functiei de bibliotecar:

```
PLIB.ERL
PMOD1.ERL
PMOD2.ERL
```

S-a creat biblioteca PLIB din fisierele PMOD1, PMOD2

2) Pentru utilizarea functiei de conversie format relocabil:

```
L80
PLIB.REL
PMOD1.ERL
PMOD2.ERL
```

Fisierele de intrare pot contine module separate rezultate din compilari/asamblari sau alte biblioteci de module; tipul lor trebuie precizat dar poate fi diferit de ERL.

Bibliotecarul nu poate prelucra module cmpilate cu optiunea $X.

Ordinea de introducere a modulelor in biblioteca este importanta deoarece linkeditorul LINKMT (ca si L80) parcurge bibliooteca o singura data extragind numai modulele ce contin simboluri referite anterior (cu optiunea /S).

Daca exista o referire la un punct de intrare dintr-un modul atunci LINKMT include tot modulul respectiv in programul creat, indiferent cite proceduri fac parte din acel modul; de aceea este bine ca procedurile destinate introducerii in biblioteci sa fie compilate ca module distincte.

Nu se recomanda modificarea bibliotecii PASLIB cu bibliotecarul.

10. UTILIZAREA DEZASAMBLORULUI PASCAL/MT+

Dezasamblorul DIS8080 trebuie sa primeasca ca intrari un fisier .PRN (rezultat dintr-o compilare cu optiunea $P) si un fiser .ERL extins (rezultat din folosirea optiunii $X la compilare); el produce un fisier de iesire continind codul simbolic 8080 generat de compilator pentru fiecare instructiune Pascal, adresele variabilelor Pascal (ca adrese relative in modul) precum si alte informatii utile pentru depanare la nivel de instructiuni masina.

Fisierul de iesire poate fi scos pe disc sau pe orice alt dispozitiv CP/M (CON:, LST:); daca nu se precizeaza iesirea atunci se listeaza la consola.

Numarul de linii afisate pe o pagina este de 66, dar poate fi modificat printr-o optiune (,L=nn) asociata fisierului de iesire in linia de comanda.

<u>Exemplu:</u>

```
PASCAL PROG $P@ X
DIS8080 PROG,LST:,L=58  sau  DIS8080 PROG,PROG.DIS
```

11. LIMBAJUL PASCAL/MT+. PARTICULARITATI SI EXTINDERI.

Limbajul Pascal/MT+ reprezinta o extindere a limbajului Pascal definit in standardul ISO si in raportul PASCAL (Jensen, Wirth).

Principalele facilitati din Pascal-ISO prezente si in Pascal/MT sint:

- Tipuri de date predefinite: REAL, INTEGER, CHAR, BOOLEAN,
  ARRAY (tablouri cu mai multe dimensiuni);
  RECORD (inregistrari definite de programator);
  referinta (Pointer) catre date de orice tip;
  FILE (fisiere de orice tip si fisiere TEXT);
  SET (multimi cu max 256 de elemente).
- Tipuri enumerate, subdomeniu si tipuri definite de utilizator;
- Proceduri si functii, care pot avea ca parametrii nume de proceduri sau functii;
- Tablouri conforme ca parametrii de proceduri: o procedura poate avea ca argument tablouri de dimensiuni diferite, dar cu elemente de acelasi tip si cu indici de acelasi tip (in acelasi subdomeniu de valori);
- Fisiere standard INPUT si OUTPUT, ca parametrii de program;
- Toate procedurile standard referitoare la fisiere in forma lor originala:RESET, REWRITE, GET, PUT, READ, WRITE, READLN, WRITELN;
- Toate instructiunile standard Pascal, inclusiv instr. WITH;
- Atributele PACK si UNPACK, dar fara nici un efect, deoarece toate structurile sint impachetate la nivel de octet;
- Procedurile standard NEW si DISPOSE de gestiune dinamica a memoriei, cu reutilizarea memoriei eliberate (memorie HEAP);
- Comentaiile pot fi incadrate de "{" si "}" sau de "(*" si "*)".

11.1 Extensii Pascal/MT+ fata de standard

1) Un nou caracter alfabetic :caracterul "@" ,folosit in numele unor rutine de bibliooteca. Deoarece standardul ISO defineste caracterul "@" ca echivalent pentru caraccterul "^" (indicator de referentiere sau de adresare indirecta) este prevazuta optiunea de compilare "@" care exclude acest caracter din multimea literelor si ii atribuie semnificatia din standard.

2) Identificatorii (numele simbolice) pot avea orice lungime, dar numai primele 8 caractere sint semnificative (celelalte fiind ignorate de compilator). In interiorul unui nume poate fi utilizat si caracterul special de subliniere ("underscore"), ignorat totusi de compilator.(tastatura TPD nu are acest caracter si de aceea nu este folosit in exemple!).

3) Sint permise constante hexazecimale, precedate de caracterul $. Cifrele hexa A,B,C,D,E,F pot fi scrise si ca a,b,c,d,e,f.

4) S-a adaugat o noua constanta de tip sir: sirul nul (' ').

5) Noi tipuri de date predefinite:

BYTE :definit ca subdomeniu 0...255, cu proprietatea ca este compatibil cu tipul CHAR.

WORD :definit ca intreg fara semn (subdomeniul 0..65535)

STRING:tablou impachetat de caractere, cae contine in primul octet lungimea efectiva a sirului; compatibil cu tipul CHAR.

Lungimea maxima implicita a unui sir (STRING) este de 80, dar la declararea

unei variabile STRING se poate specifica lungimea maxima a sirului (max 255), ca in exemplul urmator:

```
VAR NUME : STRING [25]
```

6) Atributul EXTERNAL la declararea de variabile externe, introdus inaintea numelui variabilei :

```
EXTERNAL COMM : INTEGER;
```

7) Atributul ABSOLUTE la declaratii de variabile, cu sintaxa:

```
VAR nume : ABSOLUTE [const] tip
```

unde "const" este o constanta (de obicei in hexa) care specifica adresa absoluta de memorie a variabilei declarate. Exemplu:

```
VAR BDOS : ABSOLUTE [6] WORD;
ECRAN:ABSOLUTE [$C000] ARRAY [0..23] OF ARRAY [0..79] OF CHAR;
```

Nu sint permise variabile absolute de tip STRING la adrese mai mici de 100H (datorita modului de lucru al compilatorului).

8) Operatori logici pentru variabile ne-booleene cu lungimea de unul sau de doi octeti (INTEGER, WORD, BYTE, CHAR):

& ptr. "si" logic
! sau ¦ ptr. "sau" logic
~ sau ? ptr. "nu" (complement fata de 1)

Acesti operatori au aceeasi prioritate ca operatorii analogi pentru variabile booleene.

9) Instructiunea de atribuire admite urmatoarele atribuiri intre tipuri diferite:
- unei variabile de tip BYTE i se poate atribui o vaiabila de tip CHAR sau o constanta de tip caracter intre ghilimele;
- unai variabile de tip STRING i se poate atribui rezultatul unei expresii de tip CHAR.

10) Instructiunea CASE poate avea o clauza ELSE, care specifica ce actiune are loc in cazul in care expresia de selectie este diferita de toti selectorii de caz.

Exemplu:

```
CASE CH OF
 'A': WRITELN ('A');
 'Q': WRITELN ('Q');
ELSE  WRITELN ('Altceva decit A sau Q')
END;
```

11) Un nou tip de procedura: procedura de tratare a unei intreruperi, definita astfel:

```
PROCEDURE INTERRUPT [n] nume;
```

unde "n" este nivelul de intrerupere tratat de procedura.

Procedurile de tip INTERRUPT pot fi definite numai in programul principal si nu pot avea parametrii.

12) Atributul EXTERNAL inaintea antetului de procedura (inaintea cuvintului

PROCEDURE) pentru proceduri externe, compilate separat sau scrise in limbaj de asamblare. Procedurile externe trebuie declarate inainte de prima procedura interna (locala).

13) Variabilele de tip fisier (FILE) sint asociate automat de catre cmpilator cu fisiere disc temporare, avind numele PASTMPnn si extensia .$$$ (nn=00,01,02,..). Un asemenea fisier este creat la apelarea procedurii REWRITE. In standardul Pascal variabilele fisier au asociate automat cite o zona de memorie interna.

14) O noua functie standard pentru conversie de tip :
WRD(x) transforma o variabila sau o expresie "x" de tip intreg in tipul WORD (intreg fara semn).

15) Noi proceduri si functii standard pentru:
- operatii pe siruri de caractere;
- operatii la nivel de octet si de bit;
- operatii cu fisiere secventiale si in acces direct;
- diverse alte functii si proceduri.

Aceste proceduri vor fi prezentate in detaliu mai departe.

16) Cuvintul cheie INLINE permite inserarea de instructiuni masina simbolice sau numerice intr-un text Pascal, dupa sintaxa urmatoare:

```
INLINE (arg/arg/arg/...);
```

unde fiecae din argumentele notate cu "arg" poate fi:
- un cod numeric de instructiune 8080 sau Z80, ca un caz particular de constanta Pascal (de obicei in hexazecimal);
- un cod mnemonic de instructiune 8080, precedat de un apostrof dublu si in care spatiile si virgula sint ignorate:

```
"MOV A,M/     sau  "MOVAM/
```

- orice constanta Pascal;
- orice variabila scalara din programul Pascal;
- o expresie de adresare autorelativa de forma

```
* + n    sau   * - n
```

unde "*" desemneaza adresa instructiunii curente.

<u>Exemple de utilizare INLINE:</u>

```
FUNCTON @NBDOS (FUNC:INTEGER; PARM:WORD):INTEGER;
 CONST CPM=5;
 VAR RESULT:INTEGER;
 BEGIN INLINE(
  "LHLD/FUNC/   {sau $2A/FUNC}
  "MOV C,L/     {sau $4D}
  "LHLD/PARM/   {sau $2A/PARM}
  "XCHG/        {sau $EB}
  "CALL/CPM/
  "MOV L,A/ MVI H,0/ SHLD/ RESULT);
 @BDOS:=RESULT;
END;
```

Aceasta functie exista in biblioteca PASLIB.

```
INLINE ("IN/$80/"ANI/$02/"JNZ/*-4);
```

17) Efectul procedurilor NEW si DISPOSE depinde de modul cum a fost linkeditat programul - cu sau fara fisierul FULLHEAP.ERL.

Daca nu se foloseste FULLHEAP, atunci biblioteca PASLIB contine doua rutine de gestiune a memoriei dinamice organizata ca stiva cu efectul urmator:

NEW : pune in stiva variabila dinamica adresata de parametru

DISPOSE : nu are nici un efect

Stiva pentru variabile dinamic creste dela sfirsitul zonei de date (statice) pina la stiva hardware (adresata de registrul SP).

Daca se foloseste fisierul FULLHEAP atunci memoria dinamica este organizata ca "Heap", iar efectul procedurilor este:

NEW : cauta cea mai mica zona de memorie suficienta pentru variabila dinamica specificata si aloca aceasta zona;

DISPOSE : elibereaza zona de memorie ocupata de o variabila dinamica si o intoarce in lista spatiului disponibil.

Compactarea zonelor libere adiacente din lista spatiului disponibil se face fie de procedura NEW, fie de procedura MAXAVAIL, care inapoiaza lungimea celei mai mari zone de memorie disponibile.

11.2 Proceduri si functii predefinite in Pascal/MT+

11.2.1 Proceduri de lucru cu tablouri de caractere

procedura MOVE (s,d,n);
procedura MOVELEFT (s,d,n);
procedura MOVERIGHT (s,d,n);

unde: "s" = sursa ; "d" = destinatia ; "n" = numar de caractere

Procedurile MOVE si MOVELEFT sint echivalente .

- MOVELEFT muta "n" caractere dela stinga la dreapta.
- MOVERIGHT muta "n" caractere dela dreapta la stinga

Variabilele sursa "s" si destinatie "d" pot fi de orice tip, inclusiv referinte (pointeri) sau intregi utilizati ca pointeri. Sursa si destinatia pot fi de tipuri diferite.

Parametrul "n" poate fi orice expresie intreaga cu rezultat intre 0 si 65535.

procedura FILLCHAR (d,n,c);

unde: "d" = dstinatie ; "n" = numar de caractere ; "c" = caracter

FILLCHAR umple tabloul destinatie "d" cu "n" caractere de valoare "c". Destinatia "d" este un tablou de caractere; "n" este orice expresie cu rezultat intreg; "c" este o variabila de tip CHAR sau un literal (un caracter intre ghilimele).

11.2.2 Functii si proceduri de prelucrare siruri de caractere:

functia CONCAT (s1,s2,..,sn) de tip STRING

unde: s1,s2,..sn sint fie vaiabile de tip STRING, fie constante de tip sir sau de tip caracter.

CONCAT are ca rezultat un sir de caractere obtinut prin concatenarea sirurilor sursa "s1,s2,.." si trunchiat la primele 256 de caractere.

functia COPY (s,d,n) de tip STRING

unde: "s" = sursa ; "d" = destinatia ; "n" = numar de caractere

COPY copiaza "n" caractere din sirul sursa "s" la adresa destinatie "d". Sursa "s" este un sir (STRING) iar parametrii "d" si "n" sint expresii cu rezultat intreg.

Functiile CONCAT si COPY au un singur buffer pentru sirul rezultat, de aceea in urma unor apeluri succesive ale functiei ramine in buffer numai rezultatul ultimei executii.

procedura DELETE (s,i,n);
unde: "s" = un sir de caractere ; "i" si "n" = expresii intregi
DELETE elimina "n" caractere incepind din pozitia "i" a sirului s

procedura INSERT (s,d,i);
unde : "s" = sir sau caracter sursa , "d" = sir destinatie ;
"i" = index in sirul destinatie (o expresie intreaga)
INSERT insereaza caracterul sau caracterele sursa "s" in pozitia "i" din sirul destinatie "d"

functia LENGTH(s) de tip INTEGER
unde "s" = variabila sau constanta de tip STRING.
LENGTH intoarce ca rezultat lungimea unui sir de caractere.

functia POS (p,s) de tip INTEGER
unde "p" = un caracter sau un sir de caractere (variabila sau constanta); "s" = un sir de caractere sursa.
POS intoarce ca rezultat pozitia primei aparitii a sirului sau caracterului "p" in sirul "s".

11.2.3 Functii si proceduri la nivel de octet si de bit

Toate aceste rutine opereaza numai asupra unei variabile de unul sau de doi octeti, deci de tipul CHAR, BYTE, BOOLEAN, WORD sau INTEGER, numita in continuare "variabila de baza".

procedura CLRBIT (v,b)
unde: "v" = variabila de baza ; "b" = numar de bit (0..15)
CLRBIT sterge (pune pe 0) bitul numarul "b" din variabila "v".
Numerotarea bitilor este de la dreapta (0) la stinga (15).

procedura SETBIT (v,b);
SETBIT pune pe 1 bitul numarul "b" din variabila de baza "v".

functia TSTBIT (v,b) de tip BOOLEAN
TSTBIT intoarce un rezultat TRUE (adevarat) daca bitul numarul "b" din variabila de baza "v" are valoarea 1 si un rezultat FALSE daca acest bit este 0.

functia HI (v) de tip INTEGER;
HI intoarce ca rezultat intreg octetul superior (High) din variabila de baza "v".

functia LO (v) de tip INTEGER;
LO intoarce ca rezultat intreg octetul inferior (Low) din variabila de baza "v".

functia SWAP (v) de tip INTEGER;
SWAP schimba intre ei octetul inferior si octetul superior al variabilei de baza "v" ; are rezultat de tip intreg.

functia SHR (v,m) de tip INTEGER;
unde: "v" = variabila de baza ; "m" = expresie cu rezultat intreg;
SHR deplaseaza la dreapta cu "m" biti continutul variabilei "v" introducind zerouri prin stinga ("v" de 8 sau de 16 biti).

functia SHL (v,m) de tip INTEGER;
unde: "v" = variabila de baza ; "m" = expresie cu rezultat intreg;
SHL deplaseaza la stinga cu "m" biti continutul variabilei "v" introducind zerouri prin dreapta ("v" de 8 sau de 16 biti).

11.2.4 Functii si proceduri diverse

procedura EXIT;
EXIT termina un program, o procedura sau o functie Pascal.

functia ADDR (id) de tip INTEGER;
ADDR intoarce ca rezultat adresa asociata identificatorului "id", care poate fi un nume de variabila, de procedura sau de functie, de variabila indexata sau de componenta a unei inregistrari; se poate folosi si pentru nume externe si din alte segmente. Numele "id" trebuie sa fie "vizibil" din locul unde este folosita ADDR ; de exemplu un nume local dintr-o procedura nu este vizibil din programul principal sau din alta procedura de acelasi nivel.

functia SIZEOF (id) de tip INTEGER;
unde: "id" este numele unei variabile sau unui tip de date.
SIZEOF intoarce ca rezultat numarul de octeti necesar pentru o variabila sau pentru un tip de date definit de utilizator.

functia MEMAVAIL de tip INTEGER;
MEMAVAIL intoarce ca rezultat dimensiunea celui mai mare bloc de memorie dinamica neutilizat de procedurile NEW si DISPOSE.

functia MAXAVAIL de tip INTEGER;
MAXAVAIL colecteaza resturile de memorie eliberate prin DISPOSE, combina zonele libere adiacente si intoarce lungimea celui mai mare bloc de memorie disponibil pentru alocare dinamica.

procedura WAIT (port,masca,sens);
unde "port" si "masca" sint constante cu nume sau literale;"sens" este o constanta logica (TRUE/FALSE).
WAIT este echivalenta cu urmatoarea secventa de instructiuni:

```
  IN port
  ANI masca
  J?? $-4
```

unde ?? este Z daca sens=FALSE si ??=NZ daca sens=TRUE.

functia @CMD de tip ^STRING;
@CMD intoarce adresa unui sir de caractere, care contine coada liniei de comanda prin care a fost lansat programul (deci parametrii comenzii). Acest sir incepe intotdeauna cu un blanc, care separa numele comenzii de coada comenzii.
Aceasta functie poate fi folosita o singura data, la inceputul programului, inainte de a deschide un fisier.

Exemplu de utilizare @CMD:

```
PROGRAM CMDLINE;
VAR S: STRING; PTR: ^STRING; F: FILE OF CHAR;
EXTERNAL FUNCTION @CMD:^STRING;
BEGIN
 PTR:=@CMD; S:=PTR^;
 ASSIGN (F,S); RESET(F);...
END.
```

11.2.5 Proceduri si functii pentru lucru cu fisiere

functia IORESULT de tip INTEGER;
In urma operatiilor de I/E cu fisiere, functia intreaga fara parametrii IORESULT intoarce octetul de rezultat furnizat de BDOS

Dupa deschiderea sau inchiderea unui fisier IORESULT=255 arata o eroare la

aceste operatii (terminare anormala).

Dupa o citire sau o scriere , o valoare nenula pentru IORESULT arata o terminare anormala a operatiei : sfirsit de fisier la o citire secventiala (READ,REALN,GET) sau eroare de citire/scriere in acces direct sau secvential.

Unele proceduri (OPEN,CLOSE,..) au si un parametru pentru rezultatul operatiei, parametru care primeste aceeasi valoare ca si IORESULT.

procedura ASSIGN (f,nume);

unde: "f" = variabila de tip FILE ; "nume"=nume de fisier disc sau de dispozitiv periferic CP/M (variabila sau constanta sir de caractere).

ASSIGN asociaza o variabila fisier Pascal cu un nume extern de fisier sau cu un nume de dispozitiv logic de I/E.

Fisierul "f" poate avea componente de orice tip, dar daca se asociaza cu un nume de dispozitiv atunci variabila "f" trebuie sa fie de tipul FILE OF TEXT.

Numele de dispozitive admise sint:

CON: consola la intrare si la iesire;
KBD: claviatura consolei (numai intrare) (incompatibila cu CON:)
TRM: ecranul consolei (numai iesire)
LST: imprimanta
RDR: dispozitiv de citire
PUN: dispozitiv de perforare (de scriere)

Citirea dela KBD si afisarea la TRM nu interpreteaza nici un caracter; citirea dela CON interpreteaza caracterele CR (inlocuit cu CR,LF) si BS (inlocuit cu BS,SP,BS). Afisarea la CON interpre-teaza caracterele CR (ca CR,LF) si TAB (salt la 8 spatii).

Procedura ASSIGNN trebuie apelata inainte de a deschide un fisier prin procedurile RESET sau REWRITE; in caz contrar se utilizeaza numele implicite PASTMPnn.$$$ pentru fisiere.

procedura OPEN(f,nume,r);

unde: "f" = variabila de tip FILE ; "nume" = sir de caractere ;
"v" = variabila intreaga.

OPEN deschide fisierul desemnat prin "f" si asociaza aceasta variaila cu fisierul extern specificat prin "nume"; variabila "r" contine rezultatul deschiderii (255=eroare).

OPEN(f,nume,r) are acelasi efect cu secventa:

ASSIGN(f,nume); RESET(f); r:=IORESULT;

procedura CLOSE (f,r);

unde:"f" = variabila de tip FILE ; "r" = variabila de tip INTEGER

CLOSE inchide fisierul desemnat prin variabila "f" si atribuie variabilei "r" rezultatul operatiei (255=eroare).

procedura CLOSEDEL (f,r);

CLOSEDEL inchide si apoi sterge fisierul desemnat prin variabila "f"; variabila intreaga "r" primeste rezultatul (255=eroare).

procedura PURGE(f);

PURGE sterge fisierul desemnat prin variabila fisier "f", asociata anterior cu un nume de fisier prin ASSIGN.

functia GNB (f) de tip CHAR;

GNB (Get Next Byte) citeste octetul urmator din fisierul desemnat prin variabila "f" de tip FILE OF ARRAY OF CHAR.

functia WNB(f,c) de tip BOOLEAN

WNB (Write Next Byte) scrie caracterul "c" in fisierul desemnat de variabila "f", de tip FILE OF ARRAY OF CHAR. Rezultatul lui WNB este TRUE daca s-a produs o eroare la scriere.

Se recomanda ca dimensiunea tablourilor de caractere componente ale

fisierelor exploatate prin WNB si GNB sa fie intre 128 si 4096 de octeti; cu cit este mai mare acest buffer, cu atit timpul de prelucrare a fisierului este mai mic. Viteza sporita de lucru reprezinta de altfel motivul utilizarii functiilor WNB si GNB in locul procedurilor standard PUT si GET

procedura BLOCKREAD (f,buf,r,l,blk);
procedura BLOCKWRITE (f,buf,r,l,blk);
unde : "f" = variabila de tip FILE ; "buf" = variabila ARRAY;
"r"= variabila de tip INTEGER; "l" = expresie cu rezultat intreg
"blk" = expresie cu rezultat intreg.
BLOCKREAD citeste din fisierul desemnat prin variabila "f" in tabloul "buf" (un buffer) blocul cu numarul "blk" in lungime de "l" octeti; variabila "r" primeste rezultatul operatiei.
BLOCKWRITE scrie in fisierul "f" continutul tabloului "buf" in blocul cu numarul "blk" in lungime de "l" octeti; variabila "r" contine rezultatul (eroare daca r # 0).

Daca blk=-1 atunci se citeste sau se scrie blocul urmator din fisier (acces secvential). Valoarea maxima a lui "blk" este 32767

Dimensiunea zonei tampon "buf" trebuie sa fie egala cu lungimea "l" a blocurilor din fisier si ambele trebuie sa fie multiplu de 128 (lungimea unei inregistrari disc).

procedura SEEKREAD (f,n);
SEEKREAD citeste inregistrarea numarul "n" din fisierul asociat variabilei fisier "f" in variabila tampon "f^".

procedura SEEKWRITE (f,n);
SEEKWRITE scrie din variabila tampon "f^" in inregistrarea cu numarul "n" din fisierul asociat variabilei fisier "f".

Fisierele scrise cu SEEKWRITE sint intotdeauna contigue, indife-rent de lungimea inregistrarii, ceea ce inseamna ca aceste fisiere pot fi exploatate fie secvential fie in acces direct.

Rezultatul procedurilor SEEKxxxx se afla in IORESULT.

Procedurile SEEKWRITE si SEEKREAD nu se afla in PASLIB; ele fac parte din fisierul RANDOMIO.ERL.

procedura READHEX (f,w,b);
procedura WRITEHEX (f,e,b);
unde: "f" = variabila de tip FILE OF TEXT ; "w" = variabila de orice tip ; "e" = expresie de orice tip ; "b" = intreg in 1..2
READHEX citeste din fisierul asociat variabilei "f" in variabila "w" o valoare hexazecimala de "b" octeti.
WRITEHEX scrie in fisierul asociat cu "f" valoarea expresiei "e" in hexazecimal, pe "b" octeti.

Procedurile WRITEHEX, REAHEX pot fi folosite si pentru afisarea sau citirea de numere hexazecimale la/dela consola sau la imprimanta.

11.3 Exemple de programe Pascal cu fisiere

1) Program de creare si de afisare secventiala a unui fisier de caractere folosind procedurile standard GET si PUT.

```
PROGRAM PUTGET;
 TYPE CHFILE=FILE OF CHAR;
 VAR OUTFILE : CHFILE; RESULT : INTEGER; FILENAME : STRING [16];

PROCEDURE WRITEFILE (VAR F : CHFILE);
 VAR CH : CHAR;
```

```
 BEGIN FOR CH  := '0' TO '9' DO
  BEGIN F^ := CH ; PUT (F) END;
 END; {WRITEFILE}

PROCEDURE READFILE (VAR F : CHFILE);
 VAR I : INTEGER; CH : CHAR;
 BEGIN FOR I := 0 TO 9 DO
   BEGIN CH := F^ ; GET (F) ; WRITELN (CH) END;
 END; {READFILE}

BEGIN FILENAME := 'TEST.DAT';
 ASSIGN (OUTFILE,FILENAME); REWRITE (OUTFILE);
 IF IORESULT = 255 THEN WRITELN ('Eroare la creare',FILENAME)
  ELSE BEGIN WRITEFILE (OUTFILE); CLOSE (OUTFILE,RESULT);
    IF RESULT = 255 THEN WRITELN ('Eroare la inchidere',FILENAME)
     ELSE BEGIN WRITELN ('Inchidere corecta',FILENAME);
         RESET (OUTFILE);
         IF IORESULT = 255 THEN WRITELN ('Eroare la deschidere')
          ELSE READFILE (OUTFILE)
           END;
        END;
END.
```

2) Program pentru scriere/citire in acces direct folosind proce-durile SEEKWRITE si SEEKREAD.

```
program RANDFILE;
type PERSON=record NAME:string[30];  ADDRESS:string[30] end;
var RF:file of PERSON; S:string[30];
    I:integer; ERROR:boolean; CH:char;

procedure ERRCHK;
begin ERROR:=TRUE;
 case IORESULT of
  0:begin writeln('Corect');ERROR:=FALSE end;
  1,3,5:  writeln('Citire in afara fisierului');
  2,4:    writeln('Eroare sistem');
 end;
end; {ERRCHK}

procedure READREC;
begin
 write('Numar inregistrare:'); readln(I);
 SEEKREAD(RF,I); ERRCHK;
 if ERROR then EXIT;
 writeln(RF^.NAME,'/',RF^.ADDRESS);
end; {READREC}

procedure WRITEREC;
begin
 write('Nume:');readln(S);
 RF^.NAME:=S;
 write('Adresa:');readln(S);
 RF^.ADDRESS:=S;
 write('Nr.inregistrare:'); readln(I);
 SEEKWRITE(RF,I);ERRCHK;
end; {WRITEREC}

begin {MAIN}
```

```
  write('Nume fisier:'); readln(S);
  ASSIGN(RF,S); reset(RF);
  if IORESULT=255 then
   begin  rewrite(RF); CLOSE(RF,I); reset(RF) end;
 repeat
  write('Read/Write/Quit :'); read(CH);writeln;
  case CH of
   'R':READREC;
   'W':WRITEREC;
   'Q':begin CLOSE(RF,I);EXIT end;
  else writeln ('Numai R/W/Q !');
  end;
 until CH='Q'
end.
```

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

TURBO PASCAL

1986

CUPRINS

		Pagina
1.	**UTILIZAREA SISTEMULUI TURBO PASCAL**	4
1.1	Fisiere program executabile	4
1.2	Intrarea in executie	5
2.	**ELEMENTE DE BAZA ALE LIMBAJULUI**	8
2.1	Simboluri de baza	8
2.2	Cuvinte rezervate	8
2.3	Delimitatori	8
2.4	Identificatori standard	8
2.5	Liniile de program	9
3.	**TIPURI DE DATE**	10
3.1	Clasificarea tipurilor de date	10
3.2	Tipuri simple predefinite (standard)	11
4.	**ELEMENTE DE LIMBAJ DEFINITE DE UTILIZATOR**	12
4.1	Identificatori	12
4.2	Numere	12
4.3	Siruri	12
4.4	Caractere de control	12
4.5	Comentarii	13
4.6	Directive de compilator	13
5.	**STRUCTURA GENERALA A PROGRAMELOR**	14
5.1	Antetul programului (heading)	14
5.2	Declaratii	14
6.	**EXPRESII FORMATE CU TIPURI STANDARD DE DATE**	17
6.1	Operatori	17
6.2	Specificatori de functii	18
6.3	Evaluarea expresiilor	19
7.	**INSTRUCTIUNI**	20
7.1	Instructiuni simple	20
7.2	Instructiuni structurate	21
8.	**TIPUL SCALAR ENUMERAT SI TIPUL SUBDOMENIU**	24
8.1.	Tipul scalar enumerat	24
8.2	Tipul subdomeniu	24
8.3	Conversia tipurilor	25
8.4	Verificarea domeniului	25
9.	**TIPUL SIR DE CARACTERE (STRING)**	26
9.1	Definitia tipului sir de caractere (string)	26
9.2	Expresii sir	26
9.3	Operatorul de atribuire	26
9.4	Proceduri standard pentru siruri de caractere	27
9.5	Functii standard	28
9.6	Siruri si caractere	29
10.	**STRUCTURA DE TABLOU**	30
10.1	Definirea tablourilor	30
10.2	Tablouri multidimensionale	30

10.3 Tablouri de caractere. 31
10.4 Tablouri predefinite 31

11. STRUCTURA INREGISTRARE 32
11.1 Definirea inregistrarilor. 32
11.2 Instructiunea With 33
11.3 Inregistrari cu variante 34

12. TIPUL MULTIME (SET). 35
12.1 Definirea tipului multime (set). 35
12.2 Constructori de multimi. 35
12.3 Operatii cu multimi. 35
12.4 Operatori. 36
12.5 Asignari 36

13. STRUCTURA FISIER 37
13.1 Definirea tipului fisier 37
13.2 Operatii cu fisiere. 37
13.3 Functii standard 39
13.4 Reguli de utilizare a fisierelor 39
13.5 Fisiere text 41
13.6 Fisiere fara tip 46
13.7 Controlul operatiilor de I/O 47

14. TIPURI REFERINTA (POINTER) 49
14.1 Definirea unei variabile pointer 49
14.2 Alocarea variabilelor (New). 49
14.3 Procedurile Mark si Release. 50
14.4 Utilizarea pointerilor 50
14.5 Procedura Dispose. 50
14.6 Procedura GetMem 51
14.7 Procedura FreeMem. 51
14.8 Functia MaxAvail 51

15. CONSTANTE TIPARIBILE 52
15.1 Constante tiparibile nestructurate 52
15.2 Constante tiparibile structurate 52

16. PROCEDURI SI FUNCTII 55
16.1 Parametri. 55
16.2 Anularea verificarii tipului de parametru. 56
16.3 Parametri-variabila fara tip 56
16.4 Proceduri. 57
16.5 Functii. 60
16.6 Referinte anticipate 64

17. FISIERE INCLUSE. 66

18. SISTEMUL DE REACOPERIRE. 67
18.1 Crearea subprogramelor reacoperibile 67
18.2 Reacoperiri imbricate. 68
18.3 Utilizarea eficienta a reacoperirilor. 69
18.4 Restrictii 69

19. CP/M-80. 70
19.1 Comanda eXecute. 70
19.2 Optiuni de compilare 70
19.3 Identificatori 71
19.4 Chain si eXecute 72
19.5 Reacoperiri. 72
19.6 Fisiere. 73

19.7 Variabile absolute 73
19.8 Functia Addr 74
19.9 Matrici predefinite. 74
19.10 Instructiunea With 75
19.11 Pointeri 75
19.12 Apelarea functiilor CP/M 75
19.13 Drivere I/O definite de utilizator 76
19.14 Subprograme externe. 77
19.15 Inserare de instructiuni cod masina. 77
19.16 Tratarea intreruperilor. 78
19.17 Formatul intern al datelor 78
19.18 Rezultatele functiilor 81
19.19 Structuri de memorie 83
19.20 Administrarea memoriei 83

1. UTILIZAREA SISTEMULUI TURBO PASCAL

Sistemul TURBO Pascal implementat pentru sistemele de operare PC-DOS, MS-DOS, CP/M-86 si CP/M-80 este construit astfel incit sa satisfaca cerintele oricarei categorii de utilizatori. Pentru programatori, TURBO Pascal este un instrument extrem de eficient care asigura compilari si executii de programe aproape instantanee.

TURBO Pascal urmareste structura limbajului standard Pascal, asa cum a fost definit de K. Jensen si N. Wirth. In plus fata de limbajul standard sint oferite un numar de extensii:

- variabile adrese absolute
- manipulare bit/byte
- acces direct la memoria CPU si porturi
- siruri dinamice de caractere
- ordonarea libera a sectiunilor in partea de declaratii
- un suport complet pentru facilitatile oferite de sistemul de operare
- structuri de date tip fisier
- operatii logice cu numere declarate intregi
- sistem de reacoperire (segmentare)
- inlantuire de programe cu variabilele "common"
- fisiere de date cu acces direct
- constante structurate
- functii de conversie a tipurilor de date.

In plus, pentru calculatoare compatibile IBM PC, TURBO Pascal ofera:

- culori
- grafice
- "turtlegraphics"
- ferestre
- sunet.

1.1. Fisiere progam executabile

Fisierele program executabile ale sistemului TURBO sint de tipul .COM in CP/M-80, PC-DOS/MS-DOS sau de tip .CMD daca sistemul de operare este CP/M-86.

Pe discul distribuit cu sistemul TURBO Pascal se afla urmatoarele fisiere:

TURBO. COM
Programul TURBO Pascal: compilator, editor etc. La introducerea comenzii TURBO de la terminal, acest fisier este incarcat si TURBO intra in executie.

TURBO. OVR
Fisierul de reacoperire pentru TURBO.COM (numai in versiunea CP/M-80).

TURBO. MSG
Fisier text care contine mesajele de eroare.

TINST. COM
Programul de instalare. La introducerea comenzii TINST de la terminal se executa un menu complet al procedurii de instalare. Prezenta acestui fisier si a urmatoarelor este optionala pe discul de executie.

TINST. DTA
Informatii de instalare terminal.

TINST. MSG
Mesaje pentru programul de instalare.

Fisiere. PAS
Programe Pascal exemplu.

GRAPH. P
Numai pentru versiuni IBM PC. Contin declaratii externe necesare utilizarii rutinelor grafice si "turtlegraphics" care se afla in GRAPH. BIN.

GRAPH. BIN
Numai pentru versiuni IBM PC. Contine rutinele in limbaj masina pentru grafice.

READ. ME
Contine ultimele modificari si sugestii ale sistemului TURBO.

Pe discul de executie este obligatorie prezenta numai a TURBO. COM care ocupa 30K spatiu disc (37K pentru sisteme pe 16 biti).

1.2. Lansarea in executie

TURBO Pascal este activat dupa instalare prin tastarea comenzii TURBO. Pe ecran apare urmatorul **repertoriu:**

```
Logged drive: A

Work file:
Main file:

Edit        Compile      Run          Save
Dir         Quit         Compiler  Options

Text:     0 bytes
Free: 62903 bytes
```

Repertoriul cuprinde toate comenzile disponibile atunci cind se lucreaza cu TURBO Pascal. O comanda este activata prin tastarea literei mari din numele comenzii respective. Comanda este executata imediat (nu se tasteaza <RETURN>). Daca, dupa activarea unei comenzi, se doreste reafisarea repertoriului se va tasta orice caracter care nu activeaza o comanda (ex. :<SPACE> sau <RETURN>).

Comanda **L** (Logged drive selection)

Este folosita pentru a selecta alta unitate de disc decit cea implicita.
Daca s-a tastat L se doreste anularea comenzii, se va tasta <RETURN>. Comanda L efectueaza o resetare de disc (chiar daca nu s-a selectat alta unitate de disc) si trebuie folosita ori de cite ori se schimba discurile in unitate pentru a evita erori fatale de scriere pe disc.
Numele unitatii selectate nu se afiseaza imediat; pentru a afisa menu-ul cu unitatea curenta se tasteaza <SPACE>.

Comanda **W** (Work File Selection)

Se foloseste pentru a selecta un fisier de lucru ce urmeaza a fi editat, compilat etc. Se va raspunde cu un nume legal de fisier (maximum 8 caractere, punct si tipul fisierului). Daca se omite tipul fisierului, se presupune implicit tipul .PAS. Daca este un fisier fara tip se va specifica numai numele

fisierului respectiv si punctul.

Exemplu:

PROGRAM	devine PROGRAM. PAS
PROGRAM.	nu se schimba
PROGRAM. FIS	nu se schimba.

Trebuie evitata utilizarea tipurilor .BAK, .CHN, .COM/.CMD deoarece TURBO le foloseste in scopuri speciale.

Daca fisierul specificat nu exista pe disc, apare mesajul **New File.** Daca se lucra in editare cu un fisier ce nu a fost salvat si se specifica un alt fisier de lucru, mesajul:

WorkFile X: FILENAME. TYPE not saved. Save (Y/N)?

avertizeaza ca fisierul specificat se incarca in memorie peste cel curent alterind-ul. Se raspunde Y sau N.

Comanda **M** (Main File Selection)

Comanda M poate fi utilizata pentru a defini un fisier principal, atunci cind se lucreaza cu programe care folosesc directiva de compilator $1 pentru subprograme. Fisierul principal trebuie sa contina directivele de includere a celorlalte fisiere. Se poate defini apoi fisierul de lucru ca fiind altul decit fisierul principal. In acest fel pot fi editate mai multe fisiere incluse (subprograme), lasind neschimbat numele fisierului principal.

Cind se lanseaza o compilare si fisierul de lucru este diferit de fisierul principal, fisierul curent de lucru este salvat automat, iar fisierul principal este incarcat in memorie. Daca in timpul compilarii apare o eroare , fisierul care contine eroarea devine automat fisier de lucru, care poate fi editat. Dupa corectarea erorii si reinceperea compilarii, fisierul de lucru corectat este salvat automat si fisierul principal este reincarcat.

Pentru specificarea fisierului principal se utilizeaza aceleasi reguli ca la un fisier de lucru.

Comanda **E** (Edit)

Se foloseste pentru editarea fisierului de lucru. In regim de editare, prima linie afisata pe ecran este linia de stare, care contine urmatoarele informatii:

Line n   Col n   Insert   Indent   X:FILENAME.TYP

unde:
Line n: numarul liniei curente (fata de originea fisierului)
Col n : numarul coloanei curente (fata de originea liniei)
Insert: indica mod inserare (se comuta cu CTRL/V)
Indent: indica mod auto-indent (se comuta cu comanda CTRL/QI)
X:FILENAME.TYP : nume fisier

Comenzile de editare sint cele folosite in editorul de texte WORDSTAR. Se iese din editor cu comanda CTRL/KD. Fisierul trebuie apoi salvat cu comanda S.

Comanda **C** (Compile)

Se foloseste pentru activarea compilatorului. Va fi compilat fisierul principal. Daca acesta nu este specificat se va compila fisierul de lucru. Compilarea poate fi intrerupta in orice moment prin apasarea unei taste.

Rezultatul compilarii poate fi un program rezident in memorie (implicit), un fisier .COM, sau un fisier .CHN. Alegerea se face prin selectarea uneia din optiunile de compilare.

Comanda **R** (Run)

Activeaza un program rezident in memorie, sau daca este prezenta optiunea de compilare C, se activeaza un program in cod obiect TURBO (.COM sau .CMD). Daca programul activat cu comanda R nu a fost compilat, el va fi compilat automat.

Comanda S (Save)

Se utilizeaza pentru salvarea pe disc a fisierului de lucru curent. Vechea versiune a acestuia va fi redenumita .BAK.

Comanda **D** (Directory)

Afiseaza lista de fisiere si informatii despre spatiul disponibil pe disc.

Comanda **Q** (Quit)

Se foloseste pentru iesirea din sistemul TURBO.

Comanda O (Compiler Options)

Selecteaza un menu in care se pot schimba valorile implicite ale compilatorului.

2. ELEMENTE DE BAZA ALE LIMBAJULUI

2.1. Simboluri de baza

Litere: A la Z, a la z si _ (liniuta de subliniere)

Cifre: 0-9

Caractere speciale: + - * / = ^ < > ( ) [ ] { } . , : ; ' # $

Simboluri speciale:

operatorul de asignare: :=
operatori relationali : < > <= >=
delimitator subdomeniu: . .
paranteze : (. si .) sau [si]
comentarii : (* si *) sau { si }.

2.2. Cuvinte rezervate

#ABSOLUTE	#EXTERNAL	NIL	#SHL
AND	FILE	NOT	#SHR
ARRAY	FORWARD	#OVERLAY	#STRING
BEGIN	FOR	OF	THEN
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	TO
DIV	#INLINE	PROCEDURE	UNTIL
DO	IF	PROGRAM	VAR
DOWNTO	IN	RECORD	WHILE
ELSE	LABEL	REPEAT	WITH
END	MOD	SET	#XOR

2.3. Delimitatori

Elementele limbajului trebuie separate prin cel putin un spatiu, sfirsit de linie sau comentariu.

2.4. Identificatori standard

Definesc un numar de identificatori standard de tipuri predefinite, constante, variabile, proceduri si functii. Oricare din acesti identificatori poate fi redefinit (aceasta ar putea duce insa, la pierderea facilitatii oferite de respectivul indicator si deci la confuzie).

Addr	Delay	Lenght	Release
ArcTan	Delete	Ln	Rename
Assign	EOF	Lo	Reset
Aux	EOLN	LowVideo	Rewrite
AuxInPtr	Erase	Lst	Round
AuxOutPtr	Execute	LstOutPtr	Seek

BlockRead	Exit	Mark	Sin
BlockWrite	Exp	MaxInt	SizeOf
Boolean	False	Mem	SeekEof
BufLen	FilePos	MemAvail	SeekEoln
Byte	FileSize	Move	Sqr
Chain	FillChar	New	Sqrt
Char	Flush	NormVideo	Str
Chr	Frac	Odd	Succ
Close	Getmem	Ord	Swap
ClrEOL	GotoXY	Output	Text
ClrScr	Halt	Pi	Trm
Con	HeapPtr	Port	True
ConInPtr	Hi	Pos	Trunc
ConOutPtr	IOresult	Pred	UpCase
Concat	Input	Ptr	Usr
ConstPtr	InsLine	Random	UsrInPtr
Copy	Insert	Randomize	UsrOutPtr
Cos	Int	Read	Val
CrtExit	Integer	ReadLn	Write
CrtInit	Kbd	Real	WriteLn
DelLine	KeyPresseu		

2.4. Liniile de program

Lungimea maxima a unei linii de program este de 127 caractere. Caracterele ce depasesc aceasta lungime sint ignorate de compilator.

3. TIPURI DE DATE

3.1. Clasificarea tipurilor de date

Un tip de date poate fi definit de programator (tip declarat) sau poate fi predefinit in sistem (tip standard).

Un tip de date precizeaza multimea valorilor pe care le pot avea variabilele de tipul respectiv si multimea operatiilor care pot fi efectuate asupra valorilor tipului respectiv.

In Pascal, orice variabila trebuie asociata explicit, printr-o declaratie, cu un anumit tip de date.

Specificarea unui tip de date se poate face prin urmatoarele 6 modalitati:.

a) enumerarea componentelor multimii. Se obtine ceea ce in Pascal se numeste un **tip scalar sau enumerat**. In cadrul multimii, componentele sint ordonate linear prin secventa enumerarii;

b) asimilarea cu anumite multimi de referinta, cum ar fi multimile **N** (multimea numerelor intregi) si **R** (multimea numerelor reale). Fiecareia din aceste multimi infinite ii corespunde in cadrul unei anumite implementari o submultime (finita) de valori. Tipurile standard sint predefinite si reprezinta tocmai asemenea multimi implementate. Tipurile standard din Pascal sint: **intreg, real, boolean, caracter;**

c) stabilirea unei submultimi a unui tip scalar. Deoarece un tip din categoriile a), b) reprezinta o multime ordonata de valori, putem specifica o submultime a acesteia, prin precizarea celui mai mic si celui mai mare element din submultime. Rezulta un nou tip, cu un numar finit de valori numit **tip subdomeniu.** Tipul initial devine tipul asociat subdomeniului. Pentru ca subdomeniul trebuie sa contina un numar finit de valori, tipul asociat nu poate fi tipul real.

Tipurile a), b), c) desemneaza multimi ordonate. Ele poarta numele generic de **tipuri simple**. Un tip simplu care nu este tipul real se mai numeste si tip ordinal.

d) folosirea unor operatii asupra tipurilor de date deja definite. Operatiile ce se pot folosi sint:

- indexarea cu un tip ordinal a unui tip de date deja definit (structura de tablou);
- compunerea mai multor tipuri diferite de date (structura de inregistrare);
- formarea multimii submultimilor de valori ale unui tip ordinal (structura de multime);
- insiruirea unui numar neprecizat de componente de acelasi tip (structura de fisier).

Obtinem in toate cazurile un **tip de date structurat.**

e) asocierea la un tip deja definit a tuturor referintelor la variabilele dinamice de tipul respectiv (**tipul referinta**).

f) echivalarea cu un tip deja definit.

Deci:

```
tip            ::= tip simplu|tip structurat|tip referinta
tip simplu     ::= tip ordinal|tip real
tip ordinal    ::= tip enumerat|tip intreg|tip boolean|tip caracter|tip sub-
                   domeniu
tip structurat::= tip tablou|tip inregistrare|tip multime|tip fisier
```

3.2. Tipuri simple predefinite (standard)

Tipul intreg

- numere intregi
- cuprinse intre -32768 si 32767
- ocupa 2 octeti de memorie.

Tipul byte

- subdomeniu al tipului intreg
- numere intregi cuprinse intre 0 si 255
- ocupa un octet de memorie.

Tipul real

- numere reale
- cuprinse intre 1E-38 si 1E+38 , cu o mantisa de maximum 11 cifre semnificative
- ocupa 6 octeti de memorie
- tipul real nefiind un tip ordinal trebuie mentionate urmatoarele caracteristici:
  1) functiile PRED si SUCC nu pot avea argumente reale;
  2) numerele reale nu pot fi folosite ca indecsi de tablou;
  3) numerele reale nu pot fi folosite pentru definirea tipului de baza al unui set (multime);
  4) numerele reale nu pot fi folosite pentru a controla o instructiune FOR sau CASE;
  5) nu sint admise subdomenii ale tipului real.

Tipul boolean

- o valoare booleana poate fi valoarea logica "adevarat" sau "fals" (True , False)
- False < True
- ocupa un octet de memorie.

Tipul caracter

- un caracter din setul ASCII
- ocupa un octet de memorie.

4. ELEMENTE DE LIMBAJ DEFINITE DE UTILIZATOR

4.1. Identificatori

Identificatorii sint folositi pentru a specifica etichete, constante, tipuri, variabile, proceduri si functii. Un identificator este format dintr-o litera sau liniuta de subliniere (_), urmate de orice combinatie de litere, cifre sau liniute de subliniere. Lungimea unui identificator este limitata numai de lungimea maxima a unei linii program (127 caractere).

Exemplu:

```
TURBO
Numar_persoane
Patrat
2octeti        gresit (incepe cu o cifra)
Doua Cuvinte   gresit (contine spatiu).
```

4.2. Numere

Numerele sint constante de tipul intreg sau real. Constantele intregi sint numere intregi exprimate in zecimal sau hexazecimal (constantele hexazecimale sint precedate de $).

Constantele hexazecimale sint cuprinse intre $0000 si $FFFF.

4.3. Siruri

Un sir este o secventa de caractere cuprinse intre apostrofuri:

'Acesta este un sir de caractere'.

Daca se doreste scrierea unui apostrof in interiorul unui sir de caractere se vor scrie doua apostrofuri succesive ('').

Exemplu:

```
'TURBO'
'You''ll see'
''''      sirul contine un apostrof.
```

4.4. Caractere de control

TURBO Pascal permite includerea de caractere de control in interiorul unui sir de caractere. Sint admise doua notatii pentru caracterele de control:

- simbolul # urmat de o constanta intreaga cuprinsa intre 0..255 specifica caracterul cu valoarea ASCII corespunzatoare;
- simbolul ^ urmat de un caracter, specifica caracterul de control corespunzator.

Exemplu:

```
#10     ASCII 10 zecimal (Line Feed)
```

```
#$1B    ASCII 1B hexazecimal (Escape)
^G      Control-G (Bell)
^L      Control-L (Form Feed)
^[      Control-[ (Escape).
```

Caracterele de control pot fi concatenate scriindu-le fara separatori.
Exemplu:

```
#13#10
#27^U#20
^G^G^G^G.
```

Caracterele de control pot fi concatenate cu siruri de caractere.
Exemplu:

```
 Astept sa introduceti codul!'^G^G^G'Treziti-va'#27^U
```

4.5. Comentarii

Un comentariu poate fi introdus oriunde in program, unde este legal un delimitator. Comentariile sint delimitate de **{** si **}** sau **(*** si ***)**.
Exemplu:

```
{ Acesta este un comentariu }
(* si acesta la fel *).
```

4.6. Directive compilator

O directiva compilator este introdusa ca un comentariu cu o sintaxa speciala:

{$lista-directive}

unde lista-directive cuprinde directive compilator separate prin virgule.
Exemplu:

```
{$I-}
{$R-,B+,V-}
(*$X-*).
```

Nu trebuie lasat spatiu inainte sau dupa semnul $.

5. STRUCTURA GENERALA A PROGRAMELOR

Un program Pascal este format dintr-un antet de program urmat de un bloc de program. Blocul de program este la rindul lui format dintr-o lista de declaratii si o lista de instructiuni.

```
PROGRAM identificator (lista parametri);bloc
```

unde:

```
lista parametri::= parametru{,parametru}
```

si

```
bloc::= lista declaratii
        Begin
        lista instructiuni
        End
```

5.1. Antetul programului (heading)

Antetul este optional in TURBO Pascal. Antetul contine nume programului si optional o lista de parametrii prin care programul comunica cu mediul (o secventa de identificatori separati prin virgule ,inclusa intre paranteze).

Exemplu:

```
Program Cercuri;
Program ADUN1(INPUT,OUTPUT);
Program scrie(input,printer);
```

5.2. Declaratii

Intr-un program Pascal trebuie declarate sau definite toate denumirile simbolice folosite in instructiunile dintr-un bloc. In functie de obiectele reprezentate de aceste denumiri avem cinci categorii de declaratii:

- declaratia etichetelor
- definitia constantelor
- definitia tipurilor
- declaratia variabilelor
- declaratia procedurilor si functiilor.

Spre deosebire de limbajul standard Pascal, TURBO Pascal permite ca fiecare din aceste sectiuni sa apara o data sau de mai multe ori si in orice ordine.

5.2.1. Declaratia etichetelor

Orice instructiune dintr-un program poate fi precedata de o eticheta. O eticheta este formata dintr-un nume de eticheta urmat de **:**. Inainte de folosire fiecare eticheta trebuie declarata.

Sintaxa:

```
LABEL eticheta{,eticheta};
```

Spre deosebire de Pascal standard care admite numai etichete formate din numere de maximum 4 cifre, TURBO Pascal admite si numere si identificatori pentru a fi folosite ca etichete.

Exemplu:

```
LABEL 951;
LABEL 10,eroare,gata;
```

5.2.2. Definitia constantelor

Definitia constantelor introduce identificatori sinonimi (echivalenti) unor constante, deci identificatori ce pot fi folositi in locul constantelor cu care au fost echivalati.

Sintaxa:

CONST identificator=constanta;{identificator=constanta;}

Exemplu:

```
const
 Limita=255;
 MAX=1024;
 Parola='SESAM';
```

Urmatoarele constante sint predefinite in TURBO Pascal, deci nu mai este necesar sa fie declarate:

Nume	**Tip si valoare**
Pi	Real (3.1415926536E+00)
False	Boolean (fals)
True	Boolean (adevarat)
Maxint	Intreg maxim (32767)

5.2.3. Definitia tipurilor

Are ca scop introducerea de denumiri pentru tipurile de date specificate de utilizatori.

Sintaxa:

TYPE identificator=tip;{identificator=tip;}

unde tip este specificarea unui tip de date folosind tipuri deja definite, iar identificator este numele asociat prin definitie tipului.

Exemplu:

```
Type
Numar=integer;
Culoare=(alb,rosu,verde);
Lista=array[1..10] of real;
```

5.2.4. Declaratia variabilelor

Toate variabilele folosite intr-un program Pascal trebuie declarate inainte de utilizare. Declaratia trebuie sa preceada textual orice utilizare a variabilei respective, astfel incit compilatorul sa o "cunoasca" atunci cind este folosita.

Sintaxa:

```
VAR identificator{,identificator}:tip;
   {identificator{,identificator}:tip;}
```

Domeniul in care este valabil identificatorul este blocul in care a fost declarat si oricare din blocurile imbricate.

Observatie: un bloc care se afla in interiorul altui bloc isi poate defini o alta variabila folosind acelasi identificator (variabila locala), iar variabila declarata la nivelul superior (variabila globala) devine inaccesibila.

Exemplu:

```
VAR
 Rezultat,Subtotal:Real;
 I,J,X,Y:Integer;
 Valid,Acceptat:Boolean;
 Nume,Prenume:Array[1..20] Of Char;
```

5.2.5. Declaratia procedurilor si functiilor

Declaratiile introduc identificatori pentru proceduri si functii.

Declaratia unei proceduri defineste o procedura in cadrul programului sau procedurii curente. O procedura este apelata prin intermediul instructiunii procedura si dupa terminarea sa se executa instructiunea imediat urmatoare instructiunii de apelare.

Declaratia unei functii defineste o parte de program care calculeaza si da ca rezultat o valoare. O functie este activata atunci cind se intilneste specificatorul ei intr-o expresie.

6. EXPRESII FORMATE CU TIPURI STANDARD DE DATE

Expresiile sint constructii algoritmice formate din operanzi (variabile, constante, apeluri de functii) combinati cu ajutorul operatorilor.

6.1. Operatori

1) Minus unar (minus cu un singur operand);
2) Operatorul negatie : Not
3) Operatori de multiplicare: *, /, div, mod, and, shl, shr
4) Operatori de adunare: +, -, or, xor
5) Operatori relationali: =, <>, <, >, <=, >=, in

Daca amindoi operanzii din operatiile de multiplicare sau adunare sint de tipul intreg, atunci rezultatul este de tipul intreg. Daca cel putin unul din operanzi este de tipul real, atunci rezultatul este real.

6.1.1. Operatorul minus unar

Minus unar specifica o negatie a operandului sau, care poate fi real sau integer.

6.1.2. Operatorul negatie

Inverseaza valoarea logica a operandului sau booleaan.

Not True = False
Not False = True

TURBO Pascal permite aplicarea operatorului **not** unui operand de tip intreg in care caz are loc negarea bit cu bit.

Exemplu:

Not 0 = -1
Not -15 = 14
Not $2345 = $DCBA.

6.1.3. Operatori de multiplicare

Operator	Operatia	Tipul operanzilor	Tipul rezultatului
*	inmultire	real	real
*	inmultire	intreg	intreg
*	inmultire	real, intreg	real
/	impartire	real, intreg	real
/	impartire	intreg	intreg
/	impartire	real	real
div	impartire intreaga	intreg	intreg
mod	modulo	intreg	intreg
and	si aritmetic	intreg	intreg
and	si logic	boolean	boolean
shl	shift stinga	intreg	intreg
shr	shift dreapta	intreg	intreg

Exemplu:

```
12*34=408
123/4=30.75
123 div 4 = 30
12 mod 5 = 2
True and False = False
12 and 22 = 4
2 shl 7 = 256
256 shr 7 = 2.
```

6.1.4. Operatori de adunare

Operator	Operatia	Tipul operanzilor	Tipul rezultatului
+	adunare	real	real
+	adunare	intreg	intreg
+	adunare	real,intreg	real
-	minus	real	real
-	minus	intreg	intreg
-	minus	real,intreg	real
or	sau aritmetic	intreg	intreg
or	sau logic	boolean	boolean
xor	sau exclusiv aritmetic	intreg	intreg
xor	sau exclusiv logic	boolean	boolean

Exemplu:

```
123+456=579
456-123.0=333.0
True or False = True
12 or 22 = 30
True xor False = True
12 xor 22 =26.
```

6.1.5. Operatori relationali

=	egal
<>	diferit
<	mai mic
>	mai mare
<=	mai mic sau egal
>=	mai mare sau egal.

Rezultatul este intotdeauna boolean (**adevarat** sau **fals**).

6.2. Specificatori de functii

Un specificator de functie este un identificator urmat optional de o lista de parametri (una sau mai multe variabile sau expresii separate prin virgule) inchisa intre paranteze. Aparitia unui specificator de functie determina activarea functiei respective. Daca functia nu este o functie standard predefinita, ea trebuie declarata inainte de activare.

Exemplu:

```
Writeln (Pi*(Sqr(R)))
(Max(x,y)<25) and (z>Sqrt(x*y))
Volume(Raza, Inaltime).
```

6.3. Evaluarea expresiilor

Expresiile pot contine paranteze cu scopul cunoscut al schimbarii prioritatii de executare a operatiilor.

Ordinea descrescatoare a prioritatii operatorilor in Pascal, este urmatoarea:

```
not
* , / , div , mod , and , shl , shr
+ , - , or , xor
= , <> , < , <= , >= , > , in
```

Evaluarea expresiilor respecta urmatoarele reguli:

1) Intr-o expresie cu paranteze se evalueaza intii expresiile din paranteze;

2) Intr-o expresie fara paranteze se realizeaza intii operatorii cu prioritatea cea mai mare;

3) Intr-o expresie fara paranteze si cu operatori de aceeasi prioritate, ordinea realizarii lor este de la stinga la dreapta.

7. INSTRUCTIUNI

Sectiunea de program cu instructiuni este cea care defineste actiunea pe care trebuie s-o indeplineasca programul. Pascal este un limbaj de programare secvential. Instructiunile sint executate secvential in timp, niciodata simultan. Sectiunea de instructiuni este incadrata de cuvintele rezervate **begin** si **end**. Instructiunile sint separate prin **;**. Instructiunile pot fi simple sau structurate.

7.1. Instructiuni simple

Sint instructiuni care nu contin alte instructiuni.

7.1.1. Instructiunea de atribuire (asignare)

O instructiune de atribuire este formata dintr-un identificator de variabila urmat de operatorul de atribuire **:=** si o expresie.

Atribuirea este posibila pentru orice tip de variabila (mai putin fisiere) atita timp cit variabila (sau functia) si expresia sint de acelasi tip.

Exceptie: daca variabila este de tip real, expresia poate fi de tip intreg.

Exemplu:

```
Unghi:=Unghi * Pi;
AccesOK:=False;
VolSfera:=4*Pi*R*R;
```

7.1.2 Instuctiunea procedura

Serveste pentru a activa o procedura utilizator definita anterior, sau o procedura standard predefinita.

Instructiunea este formata dintr-un identificator de procedura urmat optional de o lista de parametri (lista de variabile sau expresii separate prin virgule si incadrate de paranteze).

Cind in cursul executiei unui program este intilnita o instructiune procedurala, controlul este transferat procedurii respective, si valorile (sau adresele) posibililor parametri sint transmise procedurii. Cind se termina de executat procedura, executia programului continua de la urmatoarea instructiune.

Exemplu:

```
Find(Nume, Adresa);
Sort(Adresa);
UpperCase(Text);
UpdateFisPers(PersRecord);
```

7.1.3. Instructiunea GoTo

Este formata din cuvintul rezervat GOTO urmat de o eticheta. Reguli de utilizare a instructiunii:

1) inainte de folosire, etichetele trebuie declarate in sectiunea de declaratii a blocului in care este folosita eticheta;
2) o eticheta este valabila numai in blocul in care a fost declarata; nu este posibil saltul in sau in afara procedurii sau functiei;

7.1.4. Instructiunea vida

Este o instructiune care nu are nici un efect. Poate apare atunci cind sintaxa limbajului Pascal cere o instructiune, dar nu trebuie executat nimic.

Exemplu:

```
If A=B then
     C:=A*exp(A)
else; (* aici este o instructiune vida *)
```

7.2. Instructiuni structurate

Instructiunile structurate sint constructii compuse din mai multe instructiuni ce trebuie executate secvential (instructiuni compuse), conditionat (instructiuni conditionale) sau repetat (instructiuni repetitive).

7.2.1. Instructiunea compusa

O instructiune compusa se foloseste atunci cind trebuie executate mai multe instructiuni, desi sintaxa Pascal nu permite decit specificarea unei singure instructiuni. Instructiunea compusa este formata dintr-un numar de instructiuni separate prin **;** si incadrate de **begin** si **end.**

Exemplu:

```
If Mic > Mare then
begin
 Temp:=Mic;
 Mic:=Mare;
 Mare:=Temp;
end;
```

7.2.2. Instructiuni conditionale

O instructiune conditionala selecteaza pentru a fi executata una singura din instructiunile sale componente.

7.2.2.1. Instructiunea **If**

O instructiune se executa numai daca este adevarata o anumita conditie (expresie logica).

Observatie: **else** nu trebuie precedat de **;**.

Exemplu:

```
If (Numar<0) or (Numar > 100) then
begin
   Write ('Domeniul este 1..100, reintroduceti numarul:');
   Read(Numar);
end;
If Val > 100 then
   Dobinda:=20
else
   Dobinda:=5;
```

7.2.2.2. Instructiunea **Case**

Este formata dintr-o expresie (selectorul) si o lista de instructiuni, fiecare precedata de o eticheta care este de acelasi tip cu selectorul.

Se executa instructiunea a carei eticheta are valoarea egala cu valoarea curenta a selectorului. Daca nici una din etichete nu contine valoarea curenta a

selectorului, atunci sau nu se executa nici o instructiune, sau, optional, sint executate instructiunile care urmeaza dupa **else**.

O eticheta conditionala este formata dintr-un numar de constante sau subdomenii, separate prin virgule si urmate de **:**.

Pentru tipul selectorului sint acceptate toate tipurile scalare, mai putin tipul real.

Exemplu:

```
Case Operator of
'+':Rez:=Raspuns+Rez;
'-':Rez:=Raspuns-Rez;
'*':Rez:=Raspuns*Rez;
'/':Rez:=Raspuns/Rez;
end;
```

7.2.3. Instructiuni repetitive

Specifica executarea repetitiva a unei instructiuni. Daca numarul de repetitii este cunoscut apriori, se va utiliza instructiunea For, altfel se folosesc instructiunile While si Repeat.

7.2.3.1. Instructiunea **For**

Sintaxa:

For variabila:=val-initiala To val-finala Do instructiune

sau

For variabila:=val-initiala Downto val-finala Do instructiune

unde, val-initiala si val-finala sint expresii de acelasi tip cu variabila. Variabila poarta numele de contor al ciclului For.

Tipurile admise pentru variabila,val-initiala,val-finala sint tipurile scalare simple, standard, mai putin tipul real.

Daca val-initiala > val-finala atunci cind se foloseste clauza To, sau val-initiala < val-finala, atunci cind se foloseste clauza Downto, instructiunile din ciclu nu se executa.

Exemplu:

```
For I:=2 to 100 do If A[I]>Max then Max:=A[I];
```

7.2.3.2. Instructiunea **While**

Sintaxa:

While conditie Do instructiune;

unde conditie este o expresie cu valoare logica.

Instructiunea se executa atita timp cit conditia are valoarea **adevarat**.

Exemplu:

```
While S<0 do S:=S+A;
```

7.2.3.3. Instructiunea **Repeat**

Sintaxa:

```
Repeat
    instructiune{;instructiune}
until conditie;
```

unde conditie este o expresie de tip logic.

Se executa instructiunile din secventa (instructiunile cuprinse intre repeat si until), apoi se evalueaza conditia. Daca valoarea este adevarat executia se termina, daca valoarea este fals ciclul se repeta.

Observatie: spre deosebire de instructiunea While, instructiunea Repeat se executa cel putin o data, deoarece evaluarea expresiei conditie are loc la sfirsitul ciclului.

Exemplu:

```
Repeat
  Write(^M,'Stergeti articolul? (D/N)');
  Read(Raspuns);
until UpCase(Raspuns) in ['D','N'];
```

8. TIPUL SCALAR ENUMERAT SI TIPUL SUBDOMENIU

8.1. Tipul scalar enumerat

Tipul scalar enumerat este un tip de date de baza din Pascal. Tipul scalar enumerat reprezinta un set de valori finit si ordonat linear, notate prin identificatorii din specificatie. Ordinea valorilor este data de ordinea identificatorilor din lista. Ei sint tratati ca niste constante in textul programului in care este valabila specificarea tipului si nu pot fi reutilizati in definirea unui alt tip enumerat.

Sintaxa:

```
tip enumerat=(lista identificatori)
```

Exemplu:

```
Type
Operator=(plus,minus,inmultit,impartit)
Zi=(luni,marti,miercuri,joi,vineri,simbata,duminica)
Boolean=(fals,adevarat)
```

Variabilele de tipul Zi pot lua una din valorile luni,marti etc.

Datorita ordonarii valorilor unui tip scalar putem verifica relatia de ordine intre oricare doua valori ale aceluiasi tip folosind operatorii relationali (=, <>, <, >, <=, >=). Rezultatul aplicarii unui operator relational este o valoare logica.

Exemplu:

```
plus < minus        adevarat
vineri = joi        fals.
```

Pentru orice tip ordinal, deci si pentru tipul enumerat, exista functii standard cu un argument, ce se pot aplica valorilor tipului:

SUCC - determina succesorul argumentului

PRED - determina predecesorul argumentului

ORD - determina rangul argumentului in enumerare, stiind ca primul element are rangul 0, iar intre doua elemente consecutive rangul creste cu 1.

Exemplu:

```
Succ(marti)=miercuri
Pred(joi)=miercuri
Ord(vineri)=4.
```

8.2. Tipul subdomeniu

Pentru orice tip ordinal se poate defini un nou tip numit subdomeniu al acestui tip, precizind limitele inferioara si superioara intre care sa se afle toate valorile subdomeniului.

Sintaxa:

```
tip-subdomeniu=constanta1..constanta2
```

Exemplu:

```
Type
  Litere='A'..'Z'
  Byte=0..255
```

Un tip subdomeniu pastreaza atributele unui tip scalar asociat.

8.3. Conversia tipurilor

Functia ORD permite conversia tipurilor scalare in valori de tip intreg. Limbajul Pascal standard nu asigura o functie inversa (conversia unui intreg intr-o, valoare scalara).

In TURBO Pascal, o valoare a unui tip scalar poate fi transformata intr-o valoare a unui alt tip scalar, cu ajutorul unei facilitati de redefinire a tipului. Aceasta functie este generata prin folosirea identificatorului de tip al tipului dorit drept specificator de functie, urmat de un parametru inclus in paranteze. Parametrul poate fi o valoare a oricarui tip scalar, mai putin tipul real.

Exemplu:

```
Zi(3)=joi
Litere(14)='O'
Integer(minus)=1
Char(78)='N'
Integer('7')=55
```

8.4. Verificarea domeniului

Directiva de compilator **R** verifica variabile scalare si subdomenii. Implicit directiva este {$R-},deci fara verificare. Atunci cind se face o atribuire unei variabile scalare sau unui subdomeniu si directiva este activa ({$R+}), se verifica daca valorile atribuite apartin domeniului. Se recomanda folosirea acestei directive atita timp cit programul nu este depanat in intregime.

Exemplu:

```
Type
     Cifra=0..9;
Var
     c1,c2,c3: Cifra;
Begin
     c1:=5;          (valid)
     c2:=c1+3;       (valid; c1+3 <= 9)
     c3:=47;         (invalid, dar nu semnaleaza eroarea)
{$R+}c3:=55;         (invalid si semnaleaza eroarea)
{$R-}c3:=167;        (invalid, dar nu semnaleaza eroarea)
end.
```

9. TIPUL SIR DE CARACTERE (STRING)

Tipul sir de caractere (string) este un tip structurat, similar tipului cu structura de tablou. Diferenta intre cele doua structuri este urmatoarea: numarul de caractere dintr-un sir poate varia dinamic intre 0 si limita superioara specificata a sirului, in timp ce numarul elementelor unui tablou este fix.

9.1. Definirea tipului sir de caractere (string)

Definitia unui tip sir de caracrere trebuie sa specifice numarul maxim de caractere pe care il poate contine.

Sintaxa:

```
identificator=string[lungime-maxima]
```

unde, lungime-maxima este un intreg cuprins intre 0 si 255.

Exemplu:

```
Type
NumeFisier=string[14];
```

O variabila de tip sir de caractere ocupa lungime-maxima+1 octeti de memorie. Caracterele din cadrul unui sir sint indexate de la 1 pina la lungime-maxima.

9.2. Expresii sir

Cu variabilele de tip sir de caractere se pot construi expresii. Semnul "+" poate fi utilizat pentru a concatena siruri de caractere. Daca lungimea rezultatului este mai mare de 255 caractere, se semnaleaza eroare.

Exemplu:

```
'TURBO '+ 'Pascal' = 'TURBO Pascal'
'A' + 'B' + 'C' + 'D' = 'ABCD'
```

Operatorii relationali intorc o valoare logica (adevarat sau fals). Preced operatorul de concatenare.

Exemplu:

```
'A' < 'B'          adevarat
'A' > 'B'          fals
'TURBO'='TURBO'    adevarat
'TURBO '='TURBO'   fals
```

9.3. Operatorul de atribuire

Se utilizeaza penru a atribui valoarea unei expresii unei variabile de tip sir de caracere.

Exemplu:

```
Litere:='ABC'
```

Linie:=Litere + 'Sint primele litere ale alfabetului'

Daca lungimea maxima a unei variabile sir este depasita (asignind prea multe caractere variabilei), caracterele in plus sint trunchiate.

9.4. Proceduri standard pentru siruri de caractere

Delete

Sintaxa:
```
    Delete (St,Pos,Num);
```

unde:St = variabila string
Pos= intreg; pozitia de la care se sterg caracterele
Num= intreg; numarul de caractere care vor fi sterse.

Daca Pos este mai mare decit lungimea sirului, nu va fi sters nici un caracter. Pos poate lua valori intre 1 si 255.

Exemplu:

```
Str are valoarea 'ABCDEFG'
Delete (St,2,4) are rezultat 'AFG'
Delete (St,2,10) are rezultat 'A'
```

Insert

Sintaxa:
```
    Insert(St1,St,Pos)
```

Procedura Insert, insereaza in sirul St, sirul de caractere St1, incepind cu pozitia Pos. Daca Pos este mai mare decit lungimea sirului St, atunci St1 este concatenat la sfirsitul sirului St. Daca sirul rezultat este mai lung decit lungimea maxima a lui St, atunci caracterele in plus vor fi trunchiate. Pos trebuie sa se incadreze in limitele 1..255.

Exemplu:

```
St are valoarea 'ABCDEFG'
Insert('XX',St,3) are rezultat 'ABXXCDEFG'
```

Str

Sintaxa:
```
    Str(Val,St);
```

Procedura Str face conversia valorii numerice a lui Val intr-un sir (string) si stocheaza rezultatul in St.

Val este un parameru "write" de tipul intreg sau real, iar St este o variabila sir.

Exemplu:

```
Daca I are valoarea 1234, atunci:
    Str(I:5,St) ii da lui I valoarea    '1234'.
```

Pentru sistemele pe 8 biti: o functie ce utilizeaza procedura Str nu trebuie sa fie apelata de o expresie in cadrul instructiunilor Write si Writeln.

Val

Sintaxa:
```
    Val(St,Var,Cod);
```

Procedura Val face conversia sirului St intr-o valoare de tip intreg sau real (in functie de tipul lui Var) si stocheaza aceasta valoare in Var.

Cod este o variabila de tip intreg. Daca nu se detecteaza nici o eroare, Cod este setat pe 0, altfel Cod ia valoarea pozitiei primului caracter care a generat eroarea si valoarea lui Val este nedefinita.

Exemplu:

Daca St are valoarea '234', atunci Val(St,I,Rez) ii da lui I valoarea 234 si lui Rez valoarea 0.

Daca St are valoarea '12X' atunci Val(St,I,Rez) ii da lui I o valoare nedefinita si lui Rez valoarea 3.

9.5. Functii standard

TURBO Pascal pune la dispozitie urmatoarele functii standard:

Copy

Sintaxa:

```
Copy(St,Pos,Num);
```

Functia Copy intoarce un subsir care contine Num caractere din St, incepind cu pozitia Pos.

St este un sir de caractere, iar Num si Pos sint expresii de tip intreg.

Daca Pos este mai mare decit lungimea sirului, este intors un sir gol. Pos tebuie sa apartina subdomeniului 0..255.

Exemplu:

```
St are valoarea 'ABCDEFG'
Copy(St,3,2)   are rezultat 'CD'
Copy(St,4,10)  are rezultat 'DEFG'
Copy(St,4,2)   are rezultat 'DE'
```

Concat

Sintaxa:

```
Concat(St1,St2(,StN));
```

Functia Concat intoarce un sir care este rezultatul concatenarii argumentelor in ordinea in care au fost specificate.Pot fi concatenate oricite siruri, separate prin virgule. Lungimea sirului rezultat nu trebuie sa fie mai mare de 255.

Exemplu:

```
St1 are valoarea 'TURBO'
St2 are valoarea 'este cel mai rapid'
Concat(St1,'Pascal',St2) are rezultat
 'TURBO Pascal este cel mai rapid'
```

Length

Sintaxa:

```
Length(St);
```

Intoarce lungimea sirului St. Rezulatul este de tip intreg.

Exemplu:

```
St are valoarea '123456789'
```

Length(St) are rezultat 9.

Pos

Sintaxa:
```
    Pos(Obj,St)
```

Functia Pos inspecteaza sirul St pentru a afla pozitia in care apare pentru prima data sirul de caractere Obj. Rezultatul este de tip intreg. Daca sirul Obj nu a fost gasit, rezultatul este 0.

Exemplu:

```
St are valoarea 'ABCDEFG'
Pos('DE',St) are rezultat 4
Pos('H') are rezultat 0.
```

9.6. Siruri si caractere

Tipul sir de caractere (string) si tipul scalar char sint compatibile. Un caracter dintr-o variabila sir poate fi accesat individual prin indexare (prin adaugarea unei expresii index de tip intreg inclusa intre paranteze drepte).

Exemplu:

```
Buffer[5]
Line[Lengh(Line)-1]
Ord(Line[0])
Length(String) este acelasi lucru cu Ord(String[0]).
```

10. STRUCTURA DE TABLOU

Un tablou este un tip de date structurat format dintr-un numar fix de componente care au acelasi tip, numit tipul de **baza**. Componentele sint ordonate prin punerea lor in corespondenta cu valorile unui tip scalar numit **tipul de indexare** al tabloului. Accesul la un element component al tabloului se face folosind numele tabloului si o valoare a tipului de indexare numita **index**, ce specifica locul elementului in cadrul ansamblului.

10.1. Definirea tablourilor

Sintaxa:

```
Array[tip1] of tip2
```

unde tip1 este specificarea tipului de indexare (orice tip scalar), iar tip2 este specificarea tipului de baza.

Exemplu:

```
Type
Zi = (luni, marti, miercuri,joi, vineri, simbata, ,duminica)
Var
  Ore lucru: array[1..8] of Integer
  Saptamina: array[1..7] of Zi
```

Un component al tabloului este accesat printr-un indice incadrat in paranteze patrate, atasat identificatorului variabilei tablou.

Exemplu:

```
  Var
     Sir:array[0..19] of Char;
     x,y:char; J:0..19;
     x::= Sir[7];(* al 8-lea element din sir*)
     y::= Sir[J];(* al J-lea element din sir *)
```

10.2. Tablouri multidimensionale

Tipul de baza al unui tablou poate fi orice tip de data, chiar si un alt tablou. Acest tip de structura este denumita tablou multidimensional.

Exemplu:

```
  Var
  Matrice:array[i..j] of array[k..l] of Real;
```

Referinta la al m-lea element din tablou ce constituie, la rindul sau, al n-lea element din Matrice (adica Matrice[n]) se va scrie astfel:

Matrice[n][m].

Se poate folosi o scriere simplificata:

```
  Var
    Matrice:array[i..j,k..l] of Real;
```

si referinta la elementul din Matrice:

```
      Matrice[n,m]
```

10.3. Tablouri de caractere

Tablourile de caractere sint tablouri de un index si componente de tip scalar standard char. Pot fi considerate ca siruri de lungime constanta.

In TURBO Pascal, tablourile de caractere pot fi utilizate in expresii de tip sir de caractere, caz in care tabloul este convertit intr-un sir de lungime egala cu lungimea tabloului. Astfel tablourile pot fi comparate si manipulate in acelasi fel ca sirurile si constantele sirului pot fi atribuite tabloului de caractere atita timp cit au aceeasi lungime. Variabilele sirului si valorile calculate din expresii nu pot fi atribuite tabloului de caractere.

10.4. Tablouri predefinite

TURBO Pascal ofera tablouri predefinite de tipul Byte, numite Mem si Port, putind fi folosite pentru accesul la memoria CPU si la porturile de date.

11. STRUCTURA INREGISTRARE

O inregistrare este o structura formata dintr-un numar fix de componente, numite cimpuri. Cimpurile pot fi de tipuri diferite, fiecare cimp avind un nume (identificator de cimp).

11.1. Definirea inregistrarilor

Sintaxa:

```
Record identificator-cimp:tip{;identificator-cimp:tip}
      end;
```

unde tip poate fi sau nu identificator de tip sau descriptor de tip.

Exemplu:

```
Type
   ZileLuna = 1..31;
   Date = record
          Zi:ZileLuna;
          Luna:(ian,  feb,  mart, apr, mai, iun, iul, aug,
               sep, oct, nov, dec);
          An:1900..1999;
         end;

Var
   Datnast:Date;
```

unde: Zi, Luna, An sint identificatori de cimpuri. Un identificator de cimp trebuie sa fie unic numai in inregistrarea in care a fost definit. Un cimp este referit prin identificatorul de variabila si identificatorul de cimp separate printr-un punct.

Exemplu:

```
Datna t.Luna:=iun;
Datnast.An:=1960;
Datnast.Zi:=6;
```

Similar tipului tablou, sint acceptate asignarile intre inregistrari de acelasi tip.

Exemplu:

```
type
   Nume  = record
            Numefam:string[32];
            Pren:array[1..3] of string[16];
           end;

   Per   = record
            Pernorm, Perplus, Pernoapte : Integer;
           end;

   Date  = record
            Zi:  1..31;
```

```
                Luna:  (ian,  feb,  mart, apr, mai, iun, iul,


                        aug, sept, oct, nov, dec);
                An:  1900..1999;
               end;

      Pers   = record
                 ID:Nume;
                 Timp:Date;
               end;

      Cistig = record
                 Individual:Pers;
                 Cost:Per;
               end;

   Var
      Salariu:Cistig;
```

Avind in vedere aceste definitii, urmatoarele asignari sint legale:

```
Salariu.Cost.Perplus:=950;
Salariu.Individual.ID.Numefam:= 'Smith';
```

11.2. Instructiunea With

Folosirea inregistrarilor ca in exemplul de mai sus are drept rezultat instructiuni foarte lungi.

Cu ajutorul functiei With se pot accesa cimpuri individuale dintr-o inregistrare ca si cum ar fi simple variabile.

O instructiune With este formata din cuvintul rezervat **With** urmat de o lista de variabile-inregistrari separate prin virgule, cuvintul rezervat **do** si o instructiune.

Exemplu:

```
With Salariu do
begin
     Individual:=NouAngajat;
     Cost:=Pernorm;
end;
```

Inregistrarile pot fi imbricate in instructiunile With:

```
With Salariu,Individual,ID do
begin
     Numefam:= 'Smith';
     Pren[1]:= 'James';
end;
```

Aceasta este echivalenta cu :

```
With Salariu do With Individual do With ID do.
```

Numarul maxim de imbricari depinde de implementarea sistemului TURBO Pascal.

11.3. Inregistrari cu variante

Sintaxa tipului de date inregistrare permite existenta unei parti variante (structuri alternative de inregistrari care permit cimpurilor dintr-o inregistrare sa fie formate dintr-un numar si tip diferit de componente).

O parte varianta este formata dintr-un cimp eticheta, cu tipul definit anterior, ale carui valori determina variatia, urmat de etichete care corespund fiecarei valori posibile a cimpului eticheta.

Presupunind existenta tipului:

Origine=(cetatean, strain)

si a tipurilor Nume si Date, urmatoarea inregistrare permite cimpului Cetatenie sa aiba structuri diferite in functie de valoarea cimpului (cetatean sau strain).

Exemplu:

```
type
     Pers = record
                 Numepers:Nume;
                 Datnast:Date;
                 case Cetatenie:Origine of
                      cetatean:(Locnast:Nume);
                      strain:(Taraorigine:Nume ;
                              Dataintrarii:Date;
                              Punctintrare:Nume);
            end;
```

In aceasta definitie a unei inregistrari-variante, cimpul-eticheta este un cimp explicit care poate fi selectat si actualizat. Astfel, daca Pasager este o variabila de tipul Pers, instructiunile urmatoare sint perfect legale:

```
Pasager.Cetatenie:=cetatean;
With Pasager, Numepers do
     if Cetatenie=strain then writeln (Numefam);
```

Partea fixa a unei inregistrari trebuie sa preceada intotdeauna partea varianta. O inregistrare poate avea o singura parte varianta. Parantezele sint obligatorii, chiar daca nu inchid nimic intre ele.

12. TIPUL MULTIME (SET)

12.1. Definirea tipului multime (set)

Un tip multime (set) se defineste in raport cu un tip de baza care trebuie sa fie un tip ordinal. Dindu-se un asemenea tip de baza, tipul multime reprezinta multimea tuturor submultimilor tipului de baza (inclusiv multimea vida).

Specificarea unui tip multime:

```
type
identificator-multime = set of tip
```

Declararea unei variabile cu structura de multime:

```
var
identificator: set of tip;
```

12.2. Constructori de multimi

Multimile prelucrate de un program se pot alcatui din componentele lor folosind constructori.

De exemplu:

[5,7]

este constructorul multimii formate din intregii 5 si 7, iar

['A'..'D']

este constructorul multimii formate din caracterele 'A', 'B', 'C','D'.

Elementele unei multimi pot rezulta si din calculul unor expresii:

[a-5..a+5].

Aceasta multime formata din doua elemente depinde de valorile lui a. Daca a=3, multimea este -2..8.

Multimea [] reprezinta multimea vida.

12.3. Operatii cu multimi

Doua multimi sint egale daca si numai daca contin aceleasi elemente, indiferent de ordinea lor.

Exemplu:

[1,3,5] , [5,3,1] , [3,5,1] sint multimi egale.

Operatiile care se pot face cu multimi sint:

- reuniunea (suma) a doua multimi (A+B) este multimea ale carei elemente apartin lui A sau apartin lui B.
  Exemplu:
  Reuniunea [1,3,5,7] si [2,3,4] este [1,2,3,4,5,7].

- intersectia (produsul) a doua multimi A si B (A*B) este multimea ale carei elemente apartin si lui A si lui B.
  Exemplu:
  Intersectia [1,3,4,5,7] si [2,3,4] este [3,4].

- complementul lui B fata de A (A-B) este multimea ale carei elemente apartin lui A, dar nu apartin lui B.
  Exemplu:

Complementul [1,3,5,7] si [2,3,4] este [1,5,7].

12.4. Operatori

Ordinea efectuarii operatiilor cu multimi este data de urmatoarea ierarhie a operatorilor:

1. `*` intersectie
2. `+` reuniune
   `-` diferenta
3. `=` egalitate
   `<>` inegalitate
   `>=` "adevarat" daca elementele din a doua multime sint incluse in elementele primei multimi
   `<=` "adevarat" daca elementele primei multimi sint incluse in elementele celei de a doua multimi
   `in` test de apartenenta la multime. Al doilea operand este de tipul multime si primul operand este o expresie de acelasi tip cu tipul de baza al multimii.

Disjunctia intre doua multimi (atunci cind doua multimi nu au elemente comune) se poate exprima astfel:

A*B = []; multimea vida.

Expresiile cu multimi pot fi folosite pentru a clarifica testele. De exemplu:

```
if (Ch='T') or (Ch='U') or (Ch='R') or (Ch='B') or (Ch='O')
```

poate fi exprimat mai clar astfel:

```
Ch in ['T','U','R','B','O'].
```

Testul:

```
if (Ch >= '0') and (Ch <= '9') then...
```

poate fi exprimat:

```
if Ch in ['0'..'9'] then...
```

12.5. Asignari

Valorile rezultate din expresiile cu multimi sint asignate variabilelor tip multime cu operatorul **:=**.

Exemplu:

```
type
    ASCII = set of 0..127;
var
    NoPrint, Print, ToateCaract : ASCII;
begin
    ToateCaract:=[0..127];
    NoPrint:=[0..31,127];
    Print:=ToateCaract-NoPrint;
end;
```

13. STRUCTURA FISIER

Un fisier poate fi sau un **fisier pe disc** (informatiile sint scrise si citite pe/de pe un dispozitiv magnetic), sau un **dispozitiv logic**, asa cum sint fisierele standard Input si Output (canalele I/O standard ale sistemului-tastatura si ecranul).

Un fisier este o secventa de componente de acelasi tip, numit si **tipul de baza** al fisierului. Numarul de componente dintr-un fisier (marimea fisierului) poate fi oricit de mare. Sistemul Pascal tine evidenta acceselor la un fisier prin intermediul unui **pointer**. Ori de cite ori o componenta este scrisa sau citita in/din fisier, pointer-ul este avansat catre urmatoarea componenta. Deoarece toate componentele au lungime egala, se poate calcula pozitia oricarei componente. Astfel, pointer-ul poate fi mutat pe orice componenta din fisier.

13.1. Definirea tipului fisier

Un fisier se defineste cu cuvintele rezervate **file of**, urmate de tipul componentelor fisierului. Un identificator de fisier este declarat cu aceleasi cuvinte rezervate, urmate de identificatorul unui tip de fisier definit anterior.

Exemplu:

```
type
     NumeProdus = string [80];
     Produs = file of record
                 Nume:NumeProdus;
                 Cantitate:real;
                 Stoc:real;
                 StocMin:real;
                end;
var
     FisierProduse:Produs;
```

Tipul de baza al fisierului poate fi orice tip, mai putin tipul fisier. De exemplu:

```
var
     NomenclatorProduse:file of NumeProdus;
```

este o structura ilegala.

13.2. Operatii cu fisiere

Vom folosi identificatorul FilVar pentru a denumi un identificator de variabila fisier, declarat ca mai sus.

Assign

Sintaxa: Assign (FilVar, Str);

Str este o expresie sir de caractere care contine un nume legal de fisier. Numele de fisier este atribuit variabilei FilVar. Fisierul va fi gasit pe disc sub numele FilVar. Assign nu trebuie folosita niciodata pentru un fisier in lucru.

Rewrite

Sintaxa: Rewrite (FilVar);

Se creaza un nou fisier disc cu numele atribuit variabilei fisier FilVar. Fisierul este pregatit pentru prelucrare si pointer-ul este pozitionat la inceputul fisierului (componenta 0). Orice fisier cu acelasi nume care exista anterior, este sters. Un fisier disc creat cu Rewrite este gol initial (nu contine nici un element).

Reset

Sintaxa: Reset(FilVar);

Fisierul disc cu numele atribuit FilVar, este pregatit pentru prelucrare si pointer-ul este pozitionat pe inceput de fisier. FilVar trebuie sa existe pe disc, altfel apare o eroare I/O.

Read

Sintaxa: Read(FilVar,Var);

Var specifica una sau mai multe variabile care au tipul de baza al fisierului FilVar, separate prin virgule. Fiecare variabila este citita din fisierul disc. Dupa fiecare operatie de citire pointer-ul este avansat pe componenta urmatoare.

Write

Sintaxa: Write(FilVar,Var);

Var are aceeasi semnificatie ca mai sus. Fiecare variabila este scrisa in fisierul disc. Dupa fiecare operatie de scriere pointer-ul este avansat pe componenta urmatoare.

Seek

Sintaxa: Seek(FilVar,n);

Seek muta pointer-ul pe a n-a componenta a fisierului. n este o expresie de tip intreg. Pozitia primei componente a fisierului este 0.

Daca un fisier are extensie, este posibila cautarea unei componente dupa ultima componenta a fisierului.

Instructiunea:

```
Seek(FilVar,FileSize(FilVar));
```

plaseaza pointer-ul la sfirsitul fisierului. FileSize indica numarul de componente din fisier. Deoarece numerotarea componentelor incepe cu 0, numarul indicat este mai mic cu 1 decit numarul real de componente.

Flush

Sintaxa: Flush(FilVar)+

Flush goleste zona tampon a fisierului FilVar si astfel asigura scrierea ei pe disc, daca de la ultima actualizare a avut loc o operatie de scriere. De asemenea, Flush asigura ca urmatoarea operatie de citire va efectua o citire fizica din fisierul disc. Flush nu trebuie folosit pe un fisier inchis.

Close

Sintaxa: Close(FilVar);

Fisierul este inchis si directorul discului este actualizat, pentru a reflecta ultima stare a fisierului.

Erase

Sintaxa: Erase(FilVar);

Fisierul este sters de pe disc.

Rename

Sintaxa: Rename(FilVar,Str);

Fisierul FilVar este redenumit cu numele dat de Str. Rename nu trebuie folosit pe un fisier deschis. Programatorul trebuie sa se asigure ca nu mai exista alte fisiere cu numele Str, altfel vor fi multiple fisiere cu acelasi nume. Pentru evitarea acestui lucru se poate folosi urmatoarea functie, care intoarce True daca numele de fisier exista deja, sau False daca nu exista:

```
Type
     Nume=string[66];
     .
     .
     .
     Function Exist(NumeFis:Nume):boolean;
Var
     Fis:file;
begin
     Assign(Fis,NumeFis);
     {$I-}
     Reset(Fis);
     {$I+}
     Exist := (IOresult = 0)
end;
```

13.3. Functii standard

EOF

Sintaxa: EOF(FilVar);

Este o functie booleana care indica True daca pointer-ul este pozitionat la sfirsitul fisierului.

FilePos

Sintaxa: FilePos(FilVar);

Functia indica pozitia curenta a pointer-ului.

FileSize

Sintaxa: FileSize(FilVar);

Functia indica numarul de componente ale fisierului. Daca FileSize (FilVar) este 0, inseamna ca fisierul este vid.

13.4. Reguli de utilizare a fisierelor

Inainte de orice operatie cu un fisier, trebuie sa se atribuie un nume (Assign) variabilei-fisier respective.

Inainte de efectuarea operatiilor I/O, fisierul trebuie deschis (Reset sau Rewrite).

Un fisier disc poate fi extins numai prin adaugare de componente la sfirsitul fisierului.

Dupa terminarea operatiilor I/O, fisierul trebuie inchis (Close), altfel pot apare incidente (pierderi de informatii), deoarece directorul de disc nu a fost actualizat.

In exemplul urmator este prezentat un program care creaza un fisier numit PRODUSE.DTA si scrie 100 inregistrari de tipul Produs.

```
program InitFisProd;
const
  Nrmaxprod = 100;
type
  Numeprod = string[20];
  Produs = record
             Nume: Numeprod;
             Nrprod: integer;
             Stoc: integer;
          end;
var
  FisProd: file of Produs;
  Prodrec: Produs;
  I: integer;
begin
  Assign(FisProd, 'PRODUSE.DTA');
  Rewrite(FisProd);   {deschide  fisierul  si  sterge  orice
                        informatie}
  with Prodrec do
     begin
          Nume:= ''; Stoc:= 0;
          for I=1 to Nrmaxprod do
            begin
              Nrprod:=I;
              Write(FisProd, Prodrec);
            end;
    end;
     Close(FisProd);
  end;
```

Urmatorul program exemplifica utilizarea lui Seek in fisiere cu acces random. Programul actualizeaza fisierul FisProd creat in programul de mai sus.

```
program ActFisProd;
const
  Nrmaxprod = 100;
type
  Numeprod = string[20];
  Produs = record
             Nume: Numeprod;
             Nrprod: Integer;
             Stoc: Real;
          end;
var
  FisProd: file of Produs;
  Prodrec: Produs;
  I, Pnr: Integer;
begin
  Assign(FisProd, 'PRODUSE.DTA'); Reset(FisProd);
  ClrScr;
  Write('Introduceti numar produs (0= stop) '); Readln(Par);
  while Pnr in [1..Nrmaxprod] do
  begin
   Seek(FisProd, Pnr-1); Read(FisProd, Prodrec);
   with Prodrec do
    begin
     Write('Introduceti numele produsului (',Nume:20,') ');
     Readln(Nume);
```

```
       Write('Introduceti   numarul   de   prod.   in   stoc
              (',Stoc:20:0,')  ');
       Readln(Stoc);
       Nrprod:=Pnr;
      end;
      Seek(FisProd, Pnr-1);
      Write(FisProd, Prodrec);
      ClrScr; Writeln;
      Write('Introduceti  numarul  produsului  (0=stop)  ');
      Readln(Pnr);
    end;
    Close(FisProd);
  end;
```

13.5. Fisiere text

Componentele principale ale unui fisier text sint caracterele structurate in linii,fiecare linie fiind terminata de un indicator de sfirsit de linie (CR/LF). Un fisier text este terminat de un indicator de sfirsit de fisier (CTRL/Z).

Deoarece lungimea liniilor este variabila, pozitia unei anumite linii nu poate fi calculata, deci fisierele text nu pot fi prelucrate decit secvential.

13.5.1. Operatii pe fisiere text

O variabila fisier text este declarata prin referirea la tipul standard **Text**.

Prelucrarea fisierelor text trebuie precedata de Assign. Operatiile I/O trebuie precedate de Reset sau Rewrite.

Rewrite este folosit pentru a crea un fisier text nou, singura operatie permisa fiind adaugarea de noi componente la sfirsitul fisierului.

Reset este folosit pentru a deschide pentru citire un fisier existent, singura operatie permisa fiind citirea secventiala a fisierului.

La inchiderea unui fisier text nou, se adauga automat un indicator de sfirsit de fisier.

Introducerea/extragerea de caractere in/din fisiere text se face cu procedurile standard Read si Write. Liniile sint prelucrate cu operatorii speciali Readln, Writeln si Eoln.

Readln

Sintaxa: Readln(FilVar);
Efect: salt la inceputul liniei urmatoare.

Writeln

Sintaxa: Writeln(FilVar);
Efect: scrie un terminator de linie (CR/LF) in fisierul text.

Eoln

Sintaxa: Eoln(FilVar);
Este o functie booleana care intoarce True daca s-a ajuns la sfirsitul liniei curente. Daca EOF(FilVar) este True, atunci si Eoln(FilVar) este True.

SeekEoln

Este similara cu Eoln, numai ca sare peste spatii si TAB-uri inainte de a testa marcatorul de sfirsit de linie.

SeekEof

Sintaxa: SeekEof(FilVar);

Este similara cu EOF, numai ca sare peste spatii, TAB-uri si indicatorii de sfirsit de linie, inainte de a testa indicatorul de sfirsit de fisier.

13.5.2. Dispozitive logice

In TURBO Pascal, dispozitivele externe (terminale, imprimante, modemuri) sint considerate dispozitive logice care sint tratate ca fisiere text.

Dispozitivele logice permise sint urmatoarele:

CON: Consola. Iesirile sint transmise catre consola, dispozitiv de iesire al sistemului (CRT), iar intrarile sint obtinute de la consola, dispozitiv de intrare (tastatura). Spre deosebire de dispozitivul TRM, dispozitivul CON are zona tampon de intrare, iar operatorul are la dispozitie facilitati de editare in timpul introducerii liniilor de text.

TRM: Terminal. Liniile sint transmise catre consola dispozitiv de iesire (CRT), iar intrarile sint obtinute de la consola dispozitiv de intrare (tastatura).

KBD: Tastatura (numai input).

LST: Dispozitiv list (numai output), de obicei imprimanta.

AUX: Dispozitiv auxiliar. Pentru CP/M este RDR si PUN.

USR: Dispozitiv utilizator. Iesirile sint trimise catre rutina de iesire utilizator, iar intrarile sint obtinute de la rutina de intrare utilizator.

Dispozitivele logice pot fi accesate prin fisiere preasignate, sau pot fi asignate la variabile de fisier, exact ca un fisier disc. Nu exista nici o diferenta intre Rewrite si Reset pentru un fisier asignat la un dispozitiv logic. Close nu executa nici o functie, iar o incercare de Erase a unui astfel de fisier provoaca o eroare I/O.

Functiile standard Eof si Eoln opereaza diferit pe dispozitive logice fata de fisiere disc. Pe un fisier disc, Eof intoarce True atunci cind urmatorul caracter din fisier este CTRL/Z, sau cind s-a intilnit sfirsitul fizic de fisier, iar Eoln intoarce True atunci cind urmatorul caracter este CR sau CTRL/Z.

Pentru dispozitivele logice, Eoln si Eof opereaza pe ultimul caracter citit in loc de caracterul urmator. Eof intoarce True atunci cind ultimul caracter citit a fost CR sau CTRL/Z.

Similar, procedura Readln lucreaza diferit pe dispozitivele logice. Pe un fisier disc, Readln citeste toate caracterele pina la si inclusiv, secventa CR/LF, in timp ce pe un dispozitiv logic citeste numai pina la si inclusiv primul CR.

13.5.3. Fisiere standard

Ca o alternativa la asignarea fisierelor text la dispozitive logice asa cum a fost descris mai sus, TURBO Pascal ofera un numar de fisiere text predeclarate care au fost deja asignate la dispozitive logice specifice. Astfel, programatorul nu trebuie sa mai foloseasca procedurile Reset/Rewrite si Close, in plus facindu-se economie de cod.

Input -fisier de intrare primar. Asignat la CON: sau la TRM:

Output -fisier de iesire primar. Asignat la CON: sau la TRM:

Con -asignat la CON:

Trm -asignat la TRM:

Kbd -asignat la KBD:

Lst -asignat la LST:

Aux -asignat la AUX:

Usr -asignat la USR:.

Utilizarea procedurilor Assign, Reset, Rewrite si Close este ilegala pentru aceste fisiere.

Cind se utilizeaza procedura Read fara a specifica un identificator de fisier, procedura va introduce intotdeauna o linie, chiar daca au mai ramas de citit caractere din zona tampon, iar CRTL/Z va fi ignorat. Operatorul este fortat astfel sa termine linia cu RETURN. RETURN-ul de terminare nu apare in ecou pe ecran si intern, linia este stocata cu un CTRL/Z adaugat la sfirsit. Astfel, atunci cind sint specificate mai putine valori pe linia de intrare decit numarul de parametri din lista, orice variabile de tip Char excedentare vor fi setate pe CTRL/Z, sirurile de caractere ramin vide, iar variabilele numerice ramin nealterate.

Pentru a controla aceasta "citire fortata" se foloseste directiva de compilator **B**. Starea implicita fiind {$B+}, instructiunile Read fara un identificator fisier vor cauza introducerea unei linii de la consola. Daca directiva de compilator {$B-} este plasata la inceputul programului inainte de partea de declaratii, versiunea prescurtata a lui Read va actiona ca si cum fisierul input standard ar fi fost specificat:

Read(v1, v2,..., vn) este identic cu Read(Input, v1, v2,..., vn) si in acest caz, liniile sint introduse numai dupa ce zona tampon a fost golita.

Starea {$B-} urmeaza definitia I/O din Pascal-ul standard.

Daca nu se doreste ecou pe ecran, se va citi de la fisierul standard Kbd:

```
Read(Kbd, var);
```

Daca fisierele standard Input si Output sint folosite frecvent, ele sint selectate implicit daca nu s-a stabilit un identificator de fisier.

Lista urmatoare prezinta abrevierile pentru operatiile pe fisiere text si echivalentele lor:

```
Write(Ch)        Write(Output, Ch)
Read(Ch)         Read(Input, Ch)
Writeln          Writeln(Output)
Readln           Readln(Input)
Eof              Eof(Input)
Eoln             Eoln(Input)
```

In exemplul urmator este prezentat modul de utilizare a fisierului standard Lst pentru a lista fisierul FisProd la imprimanta:

```
program ListFisProd;
const

  Nrmaxprod = 100;
type
  Numeprod = string[20];
  Produs = record
             Nume: Numeprod; Nrprod: Integer;
             Stoc: Real;
```

```
          end;
    var
      FisProd: file of Produs;
      Prodrec: Produs; I: Integer;
    begin
      Assign(FisProd,'PRODUSE.DTA'); Reset(FisProd);
      for I := 1 to Nrmaxprod do
         begin
           Read(FisProd,Prodrec);
           with Prodrec do
           begin
           if Nume<>'' then
                Writeln(Lst,'Produs: ',Nrprod:5,' ',Nume:20,
                       'Cantitate: ',Stoc:0:0);
           end;
         end;
    Close(FisProd);
    end;
```

13.5.4. Operatii I/O pe fisiere text

Operatiile I/O pe fisiere text se executa cu procedurile standard Read, Readln, Write, Writeln care folosesc o sintaxa speciala pentru listele de parametri, ceea ce permite o flexibilitate maxima a intrarilor/iesirilor.

Parametri pot fi de tipuri diferite, procedurile I/O oferind conversii automate din/in tipul de baza Char al fisierelor text.

Daca primul parametru al unei proceduri I/O este un identificator de variabila care reprezinta un fisier text, atunci I/O actioneaza pe fisierul respectiv. Altfel, operatiile I/O vor actiona pe fisierele standard Input si Output.

Read

Asigura introducerea de caractere, siruri si date numerice.

Sintaxa:

```
     Read(Var1, Var2,..., Varn);
sau
     Read(FilVar, Var1, Var2,..., Varn);
```

unde Var1, Var2,...,Varn sint variabile de tipul Char, String, Integer sau Real.

In primul caz, variabilele sint introduse din fisierul standard Input (de obicei tastatura). In al doilea caz, datele sint introduse din fisierul text FilVar.

Cu o variabila de tipul Char, Read citeste un caracter din fisier si il atribuie variabilei respective.

Cu o variabila de tip String, Read citeste atitea caractere cite sint permise de lungimea maxima definita a sirului, daca intre timp nu a fost intilnit Eoln sau Eof.

Cu o variabila numerica, Read asteapta un sir de caractere care corespunde cu formatul unei constante numerice. Orice spatii, TAB-uri, CR sau LF care preced sirul, sint ignorate. Sirul nu trebuie sa aiba o lungime mai mare de 30 de caractere si trebuie sa fie urmat de un spatiu, TAB, CR sau CTRL/Z. Daca sirul nu corespunde formatului corect, apare o eroare I/O.

Daca se foloseste directiva de compilator {$B+} (implicita), o linie este introdusa de la consola si stocata intr-o zona tampon si citirea variabilelor foloseste aceasta zona tampon drept sursa, ceea ce permite editarea in timpul introducerii.

Sint disponibile urmatoarele facilitati de editare:

BACKSPACE si DEL
Sterge caracterul curent si se reintoarce pe cel precedent. BACKSPACE este generat de tasta marcata BS sau BACKSPACE, sau prin tastarea CTRL/H. DEL este generat de tasta DEL sau RUB sau RUBOUT.

ESC sau CTRL/X
Sterge toate caracterele introduse si reintoarcere la inceputul liniei.

CTRL/D
Cheama un caracter de la ultima linie introdusa.

CTRL/R
Cheama ultima linie introdusa.

RETURN si CTRL/M
Termina linia introdusa si stocheaza o secventa CR/LF in zona tampon.

CTRL/Z
Termina linia introdusa si stocheaza sfirsit de fisier in zona tampon.

Numarul maxim de caractere care pot fi introduse pe o linie, de la consola este 127 (implicit). Se poate micsora aceasta limita prin asignarea unui intreg in limitele 0-127 la variabila predefinita BufLen.
Exemplu:

```
Write('Nume fisier(maximum 14 caractere): ');
BufLen:=14;
Read(NumeFisier);
```

Asignarea lui BufLen afecteaza numai citirea imediat urmatoare, dupa care BufLen revine din nou la 127.

Readln

Procedura Readln este identica cu Read, cu exceptia urmatoare: dupa ce ultima variabila a fost citita , restul liniei este ignorat (toate caracterele pina la si inclusiv CR/LF).
Sintaxa:

```
    Readln(Var1, Var2,..., Varn);
sau
    Readln(FilVar, Var1, Var2,..., Varn);
```

Dupa Readln, urmatorul Read sau Readln va citi de la inceputul liniei urmatoare.
Readln poate fi chemat fara parametri:

```
    Readln
sau
    Readln(FilVar), in care caz restul liniei este ignorat.
```

Write

Procedura Write furnizeaza iesiri de caractere, siruri, valori booleene si numerice.
Sintaxa:

```
    Write(Var1, Var2,..., Varn);
sau
    Write(FilVar, Var1, Var2,..., Varn);
```

unde Var1, Var2,...,Varn (parametri de scriere) sint variabile de tipul Char, String, Boolean, Integer sau Real, urmate optional de semnul ":" si o expresie de tip intreg care defineste lungimea cimpului de iesire.

In primul caz, variabilele sint trimise catre fisierul standard Output (ecranul).

In al doilea caz, variabilele sint trimise catre fisierul text FilVar.

Formatul unui parametru de scriere depinde de tipul variabilei. In urmatoarele descrieri ale diferitelor formate si efectele lor, simbolurile:

l, m, m	desemneaza expresii de tip intreg (integer)
R	desemneaza expresii de tip real
Ch	desemneaza expresii de tip caracter
S	desemneaza expresii de tip sir (string)
B	desemneaza expresii de tip boolean.

Writeln

Se utilizeaza pentru a tipari rezultate pe linii diferite.

Sintaxa:

```
     Writeln;
sau
     Writeln(expresie{, expresie});
```

Prima forma are ca efect trecerea la o noua linie in fisierul Output. A doua forma are acelasi efect cu cel al lui Write, doar ca dupa tiparirea valorilor se marcheaza sfirsitul de linie, trecindu-se astfel la linia urmatoare in fisierul Output. Daca fisierul Output este la imprimanta, aceasta inseamna ca urmatoarea scriere se va face pe urmatorul rind.

13.6. Fisiere fara tip

Fisierele fara tip sint canale I/O utilizate primar pentru acces direct la orice fisier disc folosind o lungime de inregistrare de 128 octeti.

In operatiile de intrare/iesire pe fisiere fara tip, informatiile sint transferate direct intre fisiere disc si variabile, economisindu-se astfel spatiul necesar zonei tampon. Deoarece un fisier fara tip este compatibil cu orice fisier, utilizarea unui astfel de fisier este de preferat daca o variabila fisier este necesara doar pentru Erase, Rename sau alte operatii care nu fac I/O.

Un fisier fara tip este declarat cu cuvintul rezervat **file**. Exemplu:

```
var
     Fisdate:file;
```

BlockRead/BlockWrite

Cu exceptia lui Read, Write, Flush, sint permise toate celelalte proceduri standard care prelucreaza fisiere.

Read si Write sint inlocuite de doua proceduri de transfer de viteza inalta: BlockRead si BlockWrite.

Sintaxa de apel a acestor proceduri este:

```
     BlockRead(FilVar, Var, Recs);
     BlockWrite(FilVar, Var, Recs);
sau
     BlockRead(FilVar, Var, Recs, Result);
     BlockWrite(FilVar, Var, Recs, Result);
```

unde Var este orice variabila, iar Recs este o expresie integer care defineste numarul de inregistrari a cite 128 octeti ce va fi transferat intre fisierul disc si variabila. Parametrul optional Result intoarce numarul de inregistrari deja transferate.

Transferul incepe cu primul octet ocupat de variabila Var. Programatorul

trebuie sa se asigure ca variabila Var ocupa suficient spatiu pentru a cuprinde intreg transferul de date.

Un apel de BlockRead sau BlockWrite avanseaza pointerul cu Recs inregistrari.

Pentru a folosi aceste proceduri fisierul trebuie pregatit cu Assign si Reset sau Rewrite. Dupa prelucrare fisierul trebuie inchis (Close).

Functiile standard EOF, FilePos, FileSize si procedura standard Seek au acelasi mecanism de functionare.

Programul urmator exemplifica utilizarea fisierelor fara tip pentru a copia fisiere de orice tip:

```
program Filecopy;
const
  Recsize = 128;
  Bufsize = 200;
var
  Source, Dest : file;
  Sourcename, Destname : string[14];
  Buffer :  array[1..Recsize, 1..Bufsize] of byte;
  Recsread :integer;
begin
  Write('Copy from:   ');
  Readln(Sourcename);
  Assign(Source, Sorcename);
  Reset(Source);
  Write('     To:  ');
  Readln(Destname);
  Assign(Dest,Destname);
  Rewrite(Dest);
  repeat
     BlockRead(Source, Buffer, Bufsize, Recsread);
     BlockWrite(Dest, Buffer, Recsread);
  until (Recsread = 0) or (EOF(Source));
  Close(Source); Close(Dest);
end.
```

13.7. Controlul operatiilor I/O

Pentru controlul operatiilor I/O in timpul executiei unui program se foloseste directiva de compilator I .

Daca functia de control este inhibata ({$I-}), o eroare I/O nu provoaca oprirea programului, dar suspenda orice I/O pina cind este chemata functia standard IOresult. Aceasta functie reseteaza conditia de eroare si I/O poate fi efectuat din nou. Programatorului ii revine responsabilitatea de a lua decizia corespunzatoare in functie de tipul erorii I/O. Daca IOresult intoarce 0, aceasta indica o operatie incheiata cu succes, altfel inseamna ca a aparut o eroare I/O in tipul ultimei operatii de intrare/iesire.

Functia IOresult este de ajutor in situatiile in care oprirea unui program este rezultatul inacceptabil al unei erori I/O, ca in exemplul urmator in care programul continua sa ceara numele fisierului:

```
procedure OpenInFile;
begin
  repeat
     Write('introduceti numele fisierului ');
     Readln(Fis, Numefis);
     {$I-} Reset(Fis) {$I+};
     OK:=(IOresult = 0);
     if not OK then
      Writeln('Nu poate gasi fisierul ', Numefis);
```

```
   until OK;
 end;
```

Daca directiva I este pasiva ({$I-}), se urmeaza procedurile standard indicate mai jos pentru verificarea IOresult:

* Append	Close	Read	Seek
Assign	Erase	ReadLn	Write
BlockRead	Execute	Rename	WriteLn
BlockWrite	Flush	Reset	
Chain	* GetDir	Rewrite	
* ChDir	* MkDir	* RmDir	

* numai pentru PC-DOS/MS-DOS

14. TIPURI REFERINTA

Variabilele discutate pina acum, sint variabile statice, forma si marimea lor fiind predeterminata, ele existind atita timp cit dureaza activarea blocului de program in care au fost declarate. Totusi, programele au nevoie frecvent de structuri de date care variaza in forma si dimensiune in timpul executiei (structuri dinamice).

Astfel de variabile dinamice nu sint declarate intr-o declaratie de variabila explicita si nu pot fi referite direct prin identificatori. In schimb, se foloseste o variabila speciala care contine adresa de memorie a variabilei, adresa prin care putem referi (point) acea variabila. Aceasta variabila speciala se numeste variabila indicator (pointer).

14.1. Definirea unei variabile pointer

Un tip pointer se defineste prin simbolul ^ urmat de identificatorul de tip al variabilelor dinamice ce pot fi referite de variabila pointer de acest tip.

In exemplul urmator se arata cum poate fi declarata o inregistrare cu pointer-ii asociati. Tipul PersPointer este declarat ca un pointer la variabilele de tipul PersRecord:

```
type
  PersPointer = ^PersRecord;
  PersRecord = record
                Nume:string[50];
                Functie:string[50];
                Urmatorul:PersPointer;
               end;
var
  PrimaPers, UltimaPers, NouaPers :PersPointer;
```

Variabilele de mai sus sint variabilele pointer care refera inregistrari de tipul PersRecord.

14.2. Alocarea variabilelor (New)

Utilizarea variabilelor de tip pointer nu are sens atita timp cit nu exista unele variabile la care sa facem referinte. Variabilele noi, de orice tip, sint alocate cu procedura New. Aceasta procedura are un parametru care trebuie sa fie un pointer la variabilele de tipul pe care dorim sa-l cream.

O variabila noua de tipul PersRecord poate fi creata astfel, prin instructiunea:

```
New(PrimaPers);
```

si are efectul de a referi cu PrimaPers o inregistrare alocata dinamic, de tipul PersRecord.

Sint permise atribuiri intre variabile pointer daca au acelasi tip. Variabilele pointer de acelasi tip pot fi comparate folosind operatorii relationali = si <>.

Valoarea **nil** este compatibila cu toate tipurile de variabile indicator. Nil poate fi atribuit unei variabile pointer pentru a indica absenta unui pointer utilizabil. Nil poate fi folosit in comparatii.

Variabilele create prin procedura standard New sint stocate intr-o structu-

ra asemanatoare stivei, numita **heap**. Sistemul TURBO Pascal controleaza heap-ul prin intretinerea unui pointer, initializat la inceputul unui program cu adresa primului octet liber din memorie. La fiecare apelare a lui New, pointer-ul de heap este avansat spre partea superioara a memoriei libere cu numarul de octeti corespunzator dimensiunii noii variabile dinamice.

14.3. Procedurile Mark si Release

Atunci cind o variabila dinamica nu mai este necesara intr-un program, se utilizeaza procedurile standard Mark si Release pentru a elibera memoria alocata variabilei.

Procedura Mark asigneaza unei variabile valoarea pointer-ului de heap.

Sintaxa:

```
Mark (Var);
```

unde Var este o variabila pointer.

Procedura Release seteaza pointer-ul de heap la adresa continuta in argument.

Sintaxa:

```
Release (Var);
```

unde Var este o variabila pointer setata anterior prin Mark.

14.4. Utilizarea pointerilor

Presupunind ca am folosit procedura New pentru a crea o serie de inregistrari de tipul PersRecord si cimpul Urmatorul din fiecare inregistrare refera urmatoarea PersRecord creata, atunci urmatoarele instructiuni vor parcurge lista si vor scrie continutul fiecarei inregistrari (PrimaPers refera prima persoana din lista):

```
while PrimaPers <> nil do
with PrimaPers^ do
  begim
    writeln (Nume, 'este un ', Functie);
    PrimaPers:= Urmatorul;
  end;
```

14.5. Procedura Dispose

In locul procedurilor Mark si Release, poate fi folosita procedura standard Pascal, Dispose.

Observatie: Procedurile Mark/Release si Dispose utilizeaza mecanisme diferite de abordare a heap-ului. De aceea, ele nu vor fi niciodata mixate intr-un program (se va folosi sau Mark/Release sau Dispose).

Sintaxa:

```
Dispose(Var);
```

unde Var este o variabila pointer.

Procedura Dispose permite eliberarea numai a zonei de memorie utilizata de o anumita variabila pointer, spre deosebire de Mark/Release care elibereaza tot heap-ul de la variabila specificata in jos.

Exemplu:

```
Dispose (Var3);                 Mark (Var3)/Release (Var3);

                     Dupa                 Dupa
      Heap           Dispose              Mark/Release
     ------          ------               ------
     |Var1|          |Var1|               |Var1|
     ------          ------               ------
     |Var2|          |Var2|               |Var2|
     ------          ------               ------
     |Var3|          |    |               |    |
     ------          ------               ------
     |Var4|          |Var4|               |    |
     ------          ------               ------
     |Var5|          |Var5|               |    |
     ------          ------               ------
     |Var6|          |Var6|               |    |
     ------          ------               ------
```

14.6. Procedura GetMem

Procedura standard GetMem se foloseste pentru a aloca spatiu pe heap. Spre deosebire de New care aloca atita spatiu cit este necesar tipului referit prin argumentul sau, GetMem permite programatorului sa controleze dimensiunea spatiului alocat. Procedura GetMem se apeleaza cu doi parametri:

```
GetMem (PVar, I);
```

unde PVar este orice variabila pointer si I este o expresie de tip intreg care reprezinta numarul de octeti ce vor fi alocati.

14.7. Procedura FreeMem

Procedura FreeMem este utilizata pentru a revendica un bloc intreg de spatiu pe heap.

```
FreeMem (PVar, I);
```

unde PVar este o variabila pointer si I este o expresie de tip intreg care specifica numarul de octeti revendicati (trebuie sa fie exact numarul de octeti alocati variabilei prin GetMem).

14.8. Functia MaxAvail

Functia standard MaxAvail intoarce dimensiunea celui mai mare bloc consecutiv de spatiu liber pe heap (in octeti).

15. CONSTANTE TIPARIBILE

Constantele tiparibile sint o particularitate a sistemului TURBO. O constanta tiparibila poate fi folosita exact ca o variabila de acelasi tip. Constantele tiparibile pot fi folosite drept "variabile initializate", deoarece valoarea unei constante tiparibile este definita, in timp ce valoarea unei variabile este nedefinita atita timp cit nu a fost facuta o atribuire. Nu sint permise asignari de valori constantelor tiparibile, valorile lor fiind presupuse a fi constante.

Utilizarea constantelor tiparibile este eficienta, economisind cod atunci cind o constanta este utilizata frecvent intr-un program.

Constantele tiparibile sint definite prin specificarea valorii si a tipului.

Sintaxa:

```
identificator-constanta:tip=valoare;
```

15.1. Constante tiparibile nestructurate

O constanta tiparibila nestructurata este o constanta definita ca unul din tipurile scalare.

Exemplu:

```
const
  NrMasini:Integer=1267;
  Antet: string[7]= 'Capitol';
```

O constanta tiparibila poate fi folosita in locul unei variabile drept parametru pentru o procedura sau functie. Deoarece o constanta tiparibila este de fapt o variabila cu o valoare constanta, ea nu poate fi folosita in definirea altor constante tiparibile, exemplul de mai jos fiind ilegal:

```
const
  Min:Integer=0;
  Max:Integer=50;
type
  Domeniu:array[Min..Max] of integer;
```

15.2. Constante tiparibile structurate

Constantele structurate cuprind constante tablou, constante inregistrare si constante multime.

15.2.1. Constante tablou

O constanta tablou se defineste printr-un identificator de constanta, urmat de **:** si identificatorul de tip al unui tip tablou predefinit, urmat de **=** si valoarea constantei exprimata ca o multime de constante, separate prin virgule si inchise intre paranteze.

Exemplu:

```
type
  Stare=(Activ, Pasiv, Asteptare);
  StRep=array[Stare] of string[7];
```

```
const
  Star:StRep=('activ', 'pasiv', 'asteptare');
```

In exemplu este definita o constanta tablou (Star) care poate fi utilizata pentru a converti valori ale tipului scalar Stare in reprezentarile corespunzatoare ale sirului.

Componentele lui Star sint:

```
Star[Activ]='activ'
Star[Pasiv]='pasiv'
Star[asteptare]='asteptare'.
```

Tipul component al unei constante tablou poate fi orice tip, cu exceptia tipurilor fisier si pointer. Constantele tablou de caractere pot fi specificate ca siruri de caractere sau caractere singulare.

Astfel, definitia:

```
const
  Cifre:array[0..9] of Char=
                ('0','1','2','3','4','5','6','7','8','9');
```

poate fi exprimata mai simplu:

```
const
  Cifre:array[0..9] of Char='0123456789';
```

15.2.2. Constante tablou multidimensionale

Aceste constante sint definite prin incadrarea constantelor din fiecare dimensiune in seturi separate de paranteze, separate prin virgule. Constantele cele mai din interior corespund dimensiunii celei mai din dreapta.

Exemplu:

```
type
 Cub = array[0..1,0..1,0..1] of integer;
const
 Labirint:Cub = (((0,1),(2,3)),((4,5),(6,7)));
begin
 Writeln(Labirint[0,0,0],' = 0');
 Writeln(Labirint[0,0,1],' = 1');
 Writeln(Labirint[0,1,0],' = 2');
 Writeln(Labirint[0,1,1],' = 3');
 Writeln(Labirint[1,0,0],' = 4');
 Writeln(Labirint[1,0,1],' = 5');
 Writeln(Labirint[1,1,0],' = 6');
 Writeln(Labirint[1,1,1],' = 7');
end.
```

15.2.3. Constante inregistrare

O constanta inregistrare se defineste printr-un identificator de constanta urmat de : si de identificatorul de tip al inregistrarii definite anterior, urmat de semnul = si de o lista de cimpuri-constante separate prin ; si inchise intre paranteze.

Exemplu:

```
type
 Point      = record
                X,Y,Z:integer;
              end;
OS          = (CPM80,CPM86,MSDOS,Unix);
UI          = (CCP,MenuMaster);
```

```
Computer   = record
               SisOp: array[1..4] of OS;
               UserInterface: UI;
             end;
const
  Origo: Point    = (X:0; Y:0; Z:0;);
  SuperComp: Computer =
             (SistOp: (CPM80,CPM86,MSDOS,Unix);
             UserInterface: MenuMaster);
```

Cimpurile-constante trebuie specificate in ordinea in care apar in definitia tipului inregistrare respectiv. Daca o constanta inregistrare contine o parte varianta, programatorul trebuie sa specifice cimpurile variantei valide.

15.2.4. Constante multime

O constanta multime este formata din unul sau mai multi specificatori de elemente, separati prin virgule si inchisi intre paranteze patrate.

Un specificator de element trebuie sa fie o constanta sau o expresie de tip subdomeniu.

Exemplu:

```
type
 Mari = set of 'A'..'Z';
 Mici = set of 'a'..'z';
const
 LitereMari : Mari = ['A'..'Z'];
 Vocale     : Mici = ['a','e','i','o','u'];
```

16. PROCEDURI SI FUNCTII

Un program Pascal se compune din unul sau mai multe blocuri, fiecare dintre acestea putind fi la rindul sau alcatuit din blocuri, etc. Un astfel de bloc este o procedura, altul este o functie (numite in general subprograme). O procedura este o parte separata a unui program si ea este activata dintr-o alta zona de program printr-o declaratie de tip **procedure**. O functie este destul de asemanatoare, dar ea calculeaza si intoarce o valoare atunci cind este executata.

16.1. Parametri

Valorile pot fi trecute procedurilor si functiilor prin intermediul parametrilor. Parametrii furnizeaza un mecanism de substituire care permite ca logica subprogramului sa fie utilizata cu diferite valori initiale, producind astfel rezultate diferite.

Declaratia procedurii sau numele functiei care invoca subprogramul poate contine o lista de parametri, numiti parametrii actuali. Acestia sint furnizati parametrilor formali specificati in tipul subprogramului. Ordinea de transmitere a parametrilor este ordinea de aparitie in lista parametrilor. TURBO Pascal permite doua metode diferite de transmitere a parametrilor: prin valoare si prin referinta, care stabilesc efectul pe care il are modificarea parametrilor formali asupra parametrilor actuali.

Cind parametrii sint transmisi ca valoare, parametrul formal reprezinta o variabila locala in subprogram si modificarea parametrului formal nu are nici un efect asupra parametrului actual. Parametrul actual poate fi orice expresie, incluzind o variabila de acelasi tip cu parametrul formal corespunzator. Astfel de parametri se numesc parametrii-valoare si sint declarati in titlul subprogramului.

Exemplu:

```
procedure Exemplu (Num1, Num2: Numar; Str1, Str2: Txt);
```

unde: Numar si Txt sint tipuri definite anterior si Num1, Num2, Str1, Str2 sint parametrii formali carora le sint furnizate valorile parametrilor actuali. Tipurile parametrilor formali si parametrilor actuali trebuie sa corespunda.

De notat ca tipul parametrilor din zona de parametri trebuie sa fie specificat ca un identificator de tip definit anterior. Astfel, constructia:

```
procedure Select (Model:array[1..500] of integer);
```

nu este permisa. In schimb, tipul dorit poate fi definit in cadrul blocului in zona de definire a tipului.

Exemplu:

```
Type
  Range=array[1..500] of integer;
procedure Select(Model:Range);
```

Cind un parametru este transmis prin referinta, parametrul formal reprezinta de fapt parametrul actual pe toata durata executiei subprogramului. Orice modificare a parametrului formal antreneaza si modificarea parametrului actual, care trebuie sa fie o variabila. Parametrii transmisi prin intermediul unei referinte se numesc parametrii-variabila si sint declarati astfel:

```
procedure Exemplu(Var Num1, Num2:Numar);
```

Parametrii valoare si parametrii-variabila pot fi utilizati impreuna in aceasi procedura:

```
procedure Exemplu (Var Num1, Num2:Numar; Str1, Str2:Txt);
```

in care Num1 si Num2 sint parametri-variabila si Str1, Str2 sint parametri valoare. Toate calculele de adresa sint facute in momentul apelarii procedurii. Astfel, daca o variabila este o componenta a unei matrici, expresiile indicilor sai sint evaluate cind este apelat subprogramul.

De notat ca parametrii de tip file trebuiesc declarati mereu ca parametrii-variabila.

Cind o structura de date larga, de exemplu o matrice, trebuie sa fie transmisa unui subprogram ca parametru, folosirea unui parametru-variabila va economisi atit timp cit si memorie, ca si cum simgura informatie furnizata subprogramului este adresa parametrului actual. Un parametru valoare va solicita memorie pentru o copie suplimentara a intregii structuri de date.

16.2. Anularea verificarii tipului de parametru

In mod normal cind se folosesc parametri-variabila, parametrii formali si actuali trebuie sa se potriveasca exact. Aceasta inseamna ca subprogramele ce folosesc parametri-variabila de tip string vor accepta doar siruri de aceeasi lungime cu cea definita in subprogram. Aceasta restrictie poate fi depasita prin directiva de compilator V. Starea activa implicita {$V+} indica strict tipul de verificare, in timp ce starea pasiva {$V-} anuleaza verificarea tipului si permite trecerea sirurilor de orice lungime, indiferent de lungimea corespunzatoare parametrilor formali.

16.3. Parametri-variabila fara tip

Daca tipul unui parametru formal nu este definit, adica definirea tipului este omisa in sectiunea de parametri a titlului unui subprogram, atunci se spune ca acel parametru este fara tip. In acest caz, parametrii actuali corespunzatori pot fi de orice tip.

Parametrul formal fara tip este el insusi incompatibil cu toate tipurile si poate fi utilizat doar in cazul in care tipul datelor nu este semnificativ, ca de exemplu un parametru pentru Addr, BlockRead/Write, FillChar sau Move, ori ca specificatie de adresa a variabilelor absolute.

Exemplu:

```
procedure SchimbaVar (Var A1p, A2p; Marime:integer);
type
 A =array[1..MaxInt] of byte;
var
 A1 : A absolute A1p;
 A2 : A absolute A2p;
 Tmp: byte;
 Contor: integer;
begin
 for Contor := 1 to Marime do
 begin
  Tmp:= A1[Contor];
  A1[Contor]:=A2[Contor];
  A2[Contor]:=Tmp;
```

```
  end;
end;
```

16.4. Proceduri

O procedura poate sa fie predeclarata (standard) sau declarata de utilizator. Procedurile standard sint parti din sistemul TURBO Pascal si pot fi apelate fara nici o declaratie anterioara. O procedura declarata de utilizator poate primi numele unei proceduri standard, dar atunci acea procedura standard nu mai poate fi folosita in cadrul domeniului procedurii declarate de utilizator.

16.4.1. Declaratia procedurii.

Declaratia unei proceduri consta dintr-un titlu al procedurii urmat de un bloc, alcatuit dintr-o zona de declaratii si o zona de instructiuni.

Titlul procedurii consta din cuvintul rezervat **procedure** urmat de un identificator care devine numele procedurii, urmat optional, de o lista de parametri formali.

Exemplu:

```
procedure Test;
procedure Pozitie (X, Y:Integer);
procedure Calcul (Var Data:Matrice; Sortiment:real);
```

Zona de declaratii a unei proceduri are aceeasi forma cu cea a programului. Toti identificatorii declarati in lista de parametri formali si in zona de declaratii sint locali pentru procedura respectiva si pentru orice procedura din interiorul ei.

O procedura poate face trimitere la orice constanta, tip, variabila, procedura sau functie definita in exteriorul blocului.

Zona de instructiuni se refera la actiunea ce urmeaza a fi executata cind este apelata procedura si are forma unei instructiuni compuse. Daca identificatorul procedurii este utilizat in interiorul zonei de instructiuni a acelei proceduri, procedura va fi executata recursiv (doar in CP/M-80). De notat ca directiva A de compilator trebuie sa fie pasiva {$A-} cind se utilizeaza recursivitatea.

Urmatorul exemplu ofera un program care utilizeaza o procedura si transmite un parametru acestei proceduri. Deoarece parametrul actual furnizat unei proceduri este in unele cazuri o constanta (o expresie simpla), atunci parametrul formal trebuie sa fie un parametru valoare.

Exemplu:

```
program Cub;
var
 I:integer;
procedure Cubdesenat (x1,y1,x2,y2:integer);
var I:integer;
begin
 gotoxy (x1,y1);
   for I:=x1 to x2 do write ('-');
   for I:=y1 + 1 to y2 do
   begin
     gotoxy(x1,I); write('!');
     gotoxy(x2,I); wtite('!');
   end;
   gotoxy(x1,x2);
   for I:=x1 to x2 do write ('-');
end;             { sfirsit procedura}
```

```
begin
  ClrScr;
  forI:=1 to 5 do Cubdesenat(I*4,I*2,10*I,4*I);
  Cubdesenat (1,1,80,23);
end;
```

Deseori modificarile aduse parametrilor formali in procedura pot afecta de asemenea, parametrii actuali. In aceste cazuri se utilizeaza parametrii-variabila, ca in exemplul urmator:

```
procedure Comut(var A,B:Integer);
var
  Tmp:Integer;
begin
  Tmp:=A;
    A:=B;
    B:=Tmp;
end;
```

Cind aceasta procedura este apelata prin instructiunea:

```
Comut(I,J);
```

valorile lui I si J vor fi schimbate. Daca titlul procedurii a fost declarat ca mai jos:

```
procedure Comut(A,B:Integer);
```

adica cu parametrii valoare, atunci instructiunea Comut(I,J) nu va mai schimba I si J.

16.4.2. Proceduri standard

TURBO Pascal contine un numar de proceduri standard:

1) proceduri ce vehiculeaza siruri (descrise in capitolul 9)
2) proceduri ce manipuleaza fisiere (descrise in capitolul 13)
3) proceduri pentru alocarea variabilelor dinamice (descrise in capitolul 14)
4) proceduri de intrare/iesire (descrise in capitolul 13).

In plus fata de acestea sint accesibile urmatoarele proceduri standard, cu conditia ca terminalul la care se lucreaza sa aiba instalate comenzile asociate procedurilor:

ClrEol

Sintaxa: CrlEol;
Sterge toate caracterele de la pozitia la care se afla cursorul pina la sfirsitul liniei, fara sa mute cursorul.

ClrScr

Sintaxa: ClrScr;
Sterge ecranul si plaseaza cursorul in coltul din stinga sus. De notat ca unele terminale anuleaza si atributele video atunci cind se sterge ecranul, ceea ce poate perturba atributele selectate de utilizator.

CrtInit

Sintaxa: CrtInit;
Trimite catre subsistemul de afisare "sirul de initializare a terminalului" definit la instalarea procedurii.

CrtExit

Sintaxa: CrtExit;

Trimite catre subsistemul de afisare "sirul de resetare a terminalului" definit la instalarea procedurii.

Delay

Sintaxa: Delay(Time);

Procedura creaza o bucla ce se executa pe parcursul a Time milisecunde. Argumentul Time trebuie sa fie un intreg. Durata exacta poate varia in diverse medii de operare.

DelLine

Sintaxa: DelLine;

Sterge linia care contine cursorul si muta toate liniile urmatoare in sus cu o linie.

InsLine

Sintaxa: InsLine;

Insereaza o linie video in pozitia in care se afla cursorul. Toate liniile urmatoare sint mutate in jos cu o linie, iar linia de pe ultimul rind se pierde.

GotoXY

Sintaxa: GotoXY(Xpos, Ypos);

Muta cursorul in pozitia specificata de expresia intreaga Xpos(valoarea orizontala sau linia) si Ypos(valoarea verticala sau coloana). Coordonatele coltului din stinga sus (pozitia initiala) sint (1,1).

Exit

Sintaxa: Exit;

Iesire din blocul curent. Cind Exit se executa intr-o subrutina are ca efect iesirea din aceasta. Cind se executa in zona de instructiuni a unui program, determina terminarea acestuia. Un apel al subrutinei Exit echivaleaza cu o instructiune goto care face trimitere la o ecticheta aflata chiar inaintea instructiunii end a unui bloc.

Halt

Sintaxa: Halt;

Opreste executia programului si reda controlul sistemului de operare.

LowVideo

Sintaxa: LowVideo;

Pozitioneaza ecranul pe atributul definit ca "video cu intensitate scazuta" in procedura de instalare.

NormVideo

Sintaxa: NormVideo;

Pozitioneaza ecranul pe atributul video definit ca "video normal" in procedura de instalare.

Randomize

Sintaxa: Randomize;

Initializeaza generatorul de numere aleatoare cu o valoare oarecare.

Move

Sintaxa: Move(var1,var2,Num);

Realizeaza copierea direct in memorie a unui numar specificat de octeti. Var1 si var2 sint doua variabile de orice tip iar Num este o expresie intreaga. Procedura copiaza un bloc de Num octeti, pornind cu primul octet ocupat de var1 si mutindu-l in blocul ce incepe cu primul octet ocupat de Var2. Trebuie remarcata absenta procedurilor "Moveleft" si "Moveright". Aceasta pentru ca Move conduce in mod automat la posibilitatea de acoperire in procesul de mutare.

FillChar

Sintaxa: FillChar(Var,Num,Valoare);

Realizeaza umplerea unei zone de memorie cu o valoare data. Var este o variabila de orice tip, Num este o expresie intreaga, iar Valoare este o expresie de tip byte sau char. Astfel, Num octeti, pornind de la primul octet ocupat de Var sint umpluti cu valoarea Valoare.

16.5. Functii

Ca si procedurile, functiile sint fie standard (predeclarate), fie declarate de catre programator.

16.5.1. Declararea functiilor

Declararea unei functii consta dintr-un titlu si un bloc alcatuit dintr-o zona de declaratii urmata de o zona de instructiuni.

Titlul functiei este echivalent cu titlul procedurii, cu exceptia faptului ca titlul trebuie sa defineasca tipul rezultatului obtinut in urma executarii functiei. Aceasta se face prin adaugarea la titlu a caracterului : si a tipului:

```
function Cheie:boolean;
function Calcul (Var Valoare :Numar):real;
function Putere (X, Y:REAL):REAL;
```

Tipul rezultatului unei functii trebuie sa fie scalar, string sau pointer.

Zona de declaratii a unei functii este asemanatoare zonei de declaratii a unei proceduri.

Zona de instructiuni a unei functii este o instructiune compusa (7.2.1.). In zona de instructiuni trebuie sa apara cel putin o instructiune care sa atribuie o valoare identificatorului de functie. Ultima atribuire care a fost facuta determina rezultatul functiei. Daca identificatorul functiei apare si in zona de instructiuni proprie functiei respective, atunci functia este recursiva.

De notat ca tipul folosit in definirea tipului unei functii trebuie sa fie specificat anterior ca un identificator de tip. Astfel, constructia:

```
function Litere (Linie:LinieUtilizator):string[80];
```

nu este permisa. Trebuie ca tipului string[80] sa i se asocieze un identificator. Constructia urmatoare este corecta:

```
type
  str80 = string[80];
function Litere (Linie:LinieUtilizator):str80;
```

Datorita implementarii functiilor standard Write si Writeln, o functie ce utilizeaza procedurile Read, Readln, Write, Writeln nu poate fi apelata

niciodata de o expresie ce contine o instructiune Write sau Writeln. In sistemele pe 8 biti, acest lucru este valabil si pentru procedurile standard Str si Val.

16.5.2. Functii standard

In TURBO Pascal sint implementate urmatoarele functii standard:

- functii ce trateaza siruri
- functii referitoare la fisiere
- functii referitoare la pointeri.

Aceste functii au fost descrise in capitolele respective. In plus mai sint disponibile:

- functii aritmetice
- functii scalare
- functii de transfer
- alte functii.

16.5.2.1. Functii aritmetice

Abs

Sintaxa: Abs(Num);

Intoarce valoarea absoluta a numarului Num. Argumentul trebuie sa fie de tipul real sau intreg, iar rezultatul este de aceelasi tip cu argumentul.

ArcTan

Sintaxa: ArcTan (Num);

Calculeaza unghiul, in radiani, a carui tangenta este Num. Argumentul trebuie sa fie real sau intreg, iar rezultatul este de tip real.

Cos

Sintaxa: Cos(Num);

Calculeaza cosinusul numarului Num. Argumentul functiei este exprimat in radiani si tipul sau trebuie sa fie real sau intreg. Rezultatul este de tip real.

Exp

Sintaxa: Exp(Num);

Calculeaza valoarea exponentiala a lui Num. Argumentul functiei trebuie sa fie real sau intreg. Rezultatul este de tip real.

Frac

Sintaxa: Frac(Num);

Intoarce partea fractionara a lui Num, adica:

Frac(Num) = Num - Int(Num).

Argumentul functiei trebuie sa fie real sau intreg. Rezultatul este real.

Int

Sintaxa: Int(Num);

Intoarce partea intreaga a numarului Num, adica cel mai mare numar intreg mai mic sau egal cu Num, daca Num >= 0, sau cel mai mic numar intreg mai mare sau egal cu Num, daca Num < 0. Argumentul functiei trebuie sa fie de tip real sau intreg. Rezultatul este real.

Ln

Sintaxa: Ln(Num);
Calculeaza logaritmul natural al numarului Num. Argumentul functiei trebuie sa fie real sau intreg. Rezultatul este real.

Sin

Sintaxa: Sin(Num);
Calculeaza sinusul numarului Num. Argumentul Num este exprimat in radiani si trebuie sa fie de tip real sau intreg. Rezultatul este de tip real.

Sqr

Sintaxa: Sqr(Num);
Calculeaza patratul lui Num, adica Num * Num. Argumentul functiei trebuie sa fie real sau intreg, iar rezultatul este de tip real.

Sqrt

Sintaxa: Sqrt(Num);
Extrage radacina patrata din numarul Num. Argumentul trebuie sa fie intreg sau real, iar rezultatul este de tip real.

16.5.2.2. Functii scalare

Pred

Sintaxa: Pred(Num);
Intoarce predecesorul lui Num, daca acesta exista. Num este de orice tip scalar.

Succ

Sintaxa: Succ(Num);
Intoarce succesorul lui Num (daca exista). Num poate fi de orice tip scalar.

Odd

Sintaxa: Odd(Num);
Intoarce valoarea booleana True, daca Num este un numar impar, sau False, daca Num este un numar par. Num trebuie sa fie de tip intreg.

16.5.2.3. Functii de transfer

Functiile de transfer se utilizeaza pentru a transforma valoarea unui tip scalar in valoarea altui tip scalar.

Chr

Sintaxa: Chr(Num);
Intoarce caracterul cu valoarea ordinala data de catre expresia intreaga Num. De exemplu, Chr(65) intoarce caracterul "A".

Ord

Sintaxa: Ord(Var);
Intoarce numarul ordinal al valorii Var din setul definit de tipul Var. Ord(Var) este echivalent cu Integer(Var). Var poate fi orice tip scalar, exceptind tipul real, iar rezultatul este de tip intreg.

Trunc

Sintaxa: Trunc(Num);
Intoarce cel mai mare intreg, mai mic sau egal cu Num, daca Num < 0. Num trebuie sa fie de tip real, iar rezultatul este de tip intreg.

Round

Sintaxa: Round(Num);
Rotunjeste valoarea lui Num la cel mai apropiat numar intreg, astfel:
- daca Num >= 0, atunci Round(Num) = Trunc (Num+0.5)
- daca Num < 0, atunci Round(Num) = Trunc (Num-0.5).
Num trebuie sa fie de tip real, iar rezultatul este de tip intreg.

16.5.2.4. Alte functii standard

Hi

Sintaxa: Hi(I);
Octetul de rang inferior al rezultatului contine octetul de rang superior al valorii expresiei de tip integer I. Octetul superior al rezultatului este zero. Rezultatul este de tip intreg.

Lo

Sintaxa: Lo(I);
Intoarce octetul inferior al valorii expresiei intregi I, cu octetul superior fortat pe zero. Rezultatul este de tip intreg.

KeyPressed

Sintaxa: KeyPressed;
Intoarce valoarea True daca a fost actionata o tasta de la consola si False, daca nu a fost apasata nici o tasta. Rezultatul se obtine prin apelarea rutinei de stare a consolei sistemului de operare.

Random

Sintaxa: Random;
Intoarce un numar aleator >= 0 si < 1. Rezultatul este de tip real.

Random(Num)

Sintaxa: Random(Num);
Intoarce un numar aleator >= 0 si < Num. Num si numarul aleator rezultat sint de tipul intreg.

ParamCount

Sintaxa: ParamCount;
Este o functie de tip intreg. Intoarce numarul de parametri transmisi programului in bufferul liniei de comanda. Caracterele de tabulare si spatiu sint folosite drept separatori.

ParamStr

Sintaxa: ParamStr(N);
Este o functie de tip string. Intoarce al N-lea parametru din bufferul liniei de comanda.

SizeOf

Sintaxa: SizeOf(Nume);

Intoarce numarul de octeti ocupati in memorie de catre variabila sau tipul Nume. Rezultatul este de tip intreg.

Swap

Sintaxa: Swap(Num);

Functia Swap schimba intre ei octetii inferior si superior ai argumentului Num, valoarea rezultatului fiind de tip intreg.

Exemplu: Swap($1234) are rezultat $3412 (valoare hexa).

UpCase

Sintaxa: UpCase(Ch);

Intoarce echivalentul in majuscule al argumentului Ch, care trebuie sa fie de tip Char. Daca nu exista nici un echivalent in majuscule, argumentul este intors neschimbat.

16.6. Referinte anticipate

Un subprogram este declarat anticipat atunci cind titlul lui este separat de bloc. Acest titlu este scris exact ca orice titlu de subprogram, dar se termina prin cuvintul rezervat **forward**. Blocul urmeaza mai jos in cadrul aceleiasi zone de declaratii. De notat ca blocul este initiat printr-o copie a titlului in care se specifica doar numele, fara parametri, tipuri etc.

Exemplu:

```
program Test22;
var  X:integer;
function Sus(Var I:integer):integer;forward;
function Jos(Var I:integer):integer;
begin
 I:=I div 2;writeln(I);
 if I <> 1 then I:= Sus(I);
end;
function Sus;
begin
 while I mod 2 <> 0 do
 begin
  I:=I*3+1; writeln(I);
 end;
 I:=Jos(I);
end;
begin
 write ('Introduceti orice numar intreg: ');
 readln(X);
 X:=Sus(X);
 write ('OK. Programul s-a oprit.');
end;
```

Programul de mai sus este o versiune mai complicata a urmatorului program:

```
program Test22;
var X:integer;
begin
 write ('Introduceti orice numar intreg:');
 readln(X);
```

```
    while X <> 1 do
     begin
      if X mod 2 = 0 then X:=X div 2
                     else X:=X*3+1;
      writeln(X);
     end;
    write ('OK. Programul s-a oprit');
end.
```

17. FISIERE INCLUSE

Deoarece editorul sistemului TURBO executa operatia de editare numai in memorie, marimea codului sursa prelucrat este limitata. Directiva de compilator I poate fi folosita pentru a inlatura acest neajuns. Cu ajutorul acestei directive, codul sursa poate fi impartit in mai multe sectiuni ce vor fi concatenate in timpul compilarii. Facilitatea de includere aduce unui program un plus de claritate, deoarece subprogramele utilizate frecvent, gata testate, pot fi pastrate ca biblioteci de fisiere, de unde pot fi extrase si incluse in orice program.

Sintaxa: {$I nume-fisier}

unde, nume-fisier este un nume legal de fisier.
Daca nu este specificat tipul fisierului, implicit va fi considerata extensia PAS. Directiva trebuie specificata singura pe o linie. Daca fisierul nu are tipul specificat, trebuie lasat un spatiu intre numele fisierului si paranteza de sfirsit.

Exemplu:

```
{$IFis1.pas}
{$I Fis2.mod}
{$IFis3 }.
```

Fisierele incluse nu pot fi imbricate. Un fisier inclus nu poate include la rindul sau un alt fisier.

18. SISTEMUL DE REACOPERIRE

Sistemul de reacoperire (segmentare), permite crearea unor programe cu mult mai mari decit ar putea incape in memoria calculatorului. Tehnica consta in a stringe un numar de subprograme intr-unul sau mai multe fisiere separate de programul principal, care vor fi incarcate automat unul cite unul in aceeasi zona de memorie (zona de reacoperire). Zona de reacoperire este stabilita astfel incit sa poata contine cea mai mare procedura din fisierul de reacoperire.

18.1. Crearea subprogramelor reacoperibile

Subprogramele reacoperibile sint create automat prin simpla adaugare a cuvintului **overlay** in declaratia procedurii sau functiei:

```
overlay procedure Initializare;
si
overlay function CalculMat:Calcul;
```

Cind compilatorul intilneste o astfel de declaratie, codul sursa nu mai este transmis fisierului program principal, ci unui fisier de reacoperire separat. Numele acestui fisier va fi acelasi cu al programului principal de tip un numar ce desemneaza grupul de reacoperire (000-099). Subprogramele reacoperibile consecutive vor fi grupate impreuna.

Exemplu:

```
overlay procedure Unu;
begin
.
.
.
end;
overlay procedure Doi;
begin
.
.
.
end;
overlay procedure Trei;
begin
.
.
.
end;
```

Aceste trei proceduri vor fi grupate impreuna si plasate in aacelasi fisier de reacoperire. Daca este primul grup de reacoperire din program, fisierul de reacoperiri va avea numarul .000.

Cele trei proceduri din urmatorul exemplu vor fi plasate in fisiere de reacoperire consecutive, .000 si .001, datorita declaratiei unei proceduri care nu este reacoperibila (procedura Count).

Exemplu:

```
overlay procedure Unu;
begin
```

```
.
.
.
end;
overlay procedure Doi;
begin
.
.
.
end;
proocedure Count;
begin
.
.
.
end;
overlay procedure Trei;
begin
.
.
.
end;
```

In programul principal va fi rezervata cite o zona de reacoperire pentru fiecare grup.

18.2. Reacoperiri imbricate

Subprogramele reacoperibile pot fi imbricate.
Exemplu:

```
program overlay Demo;
.
.
.
overlay procedure Unu;
begin
.
.
.
end;
overlay procedure Doi;
      overlay procedure Trei;
      begin
      .
      .
      .
      end;
begin
.
.
.
end;
.
.
.
```

In acest exemplu au fost create doua fisiere de reacoperire.

Fisierul .000 contine procedurile Unu si Doi si se rezerva o zona de reacoperire in programul principal. Fisierul .001 contine procedura Trei care este locala procedurii Doi si se creaza o zona de reacoperire in codul procedurii Doi.

18.3. Utilizarea eficienta a reacoperirilor

Tehnica reacoperirii produce extra cod pentru gestiunea zonelor de reacoperire si necesita accese la disc in timpul executiei.

Pentru a nu se micsora viteza de executie, un subprogram reacoperibil nu trebuie apelat foarte des, sau, daca este apelat foarte des, atunci ar trebui sa se apeleze direct (fara apeluri catre alte subprograme).

Pentru utilizarea unui spatiu cit mai mic in programul principal, un grup de reacoperire trebuie sa contina cit mai multe subprograme individuale.

18.4. Restrictii

- Subprogramele reacoperibile din acelasi grup utilizeaza aceeasi zona de memorie si deci nu pot fi prezente simultan si nu trebuie sa se apeleze unul pe altul. In consecinta, pot utiliza aceeasi zona de date, ceea ce economiseste de asemenea spatiul de memorie (numai versiunea CP/M-80).
- Subprogramele reacoperibile nu pot fi declarate inainte.
- Subprogramele reacoperibile nu pot fi recursive. Aceasta restrictie poate fi inlaturata prin declararea unui program recursiv care apoi apeleaza un subprogram reacoperibil.

19. CP/M-80

In acest capitol sint descrise caracteristicile limbajului TURBO Pascal specifice implementarii CP/M-80 pe 8 biti.

19.1. Comanda eXecute

Este o comanda suplimentara a menu-ului principal al sistemului TURBO. Ea permite executia altor programe din interiorul sistemului TURBO Pascal, de exemplu copiere de programe, procesoare de texte, de fapt orice program executabil sub sistemul de operare. Cind se tasteaza **X** pe ecran apare:

```
Command : _
```

Se introduce numele oricarui program care se va incarca si va fi executat normal. Dupa iesirea din programul respectiv controlul este transferat sistemului TURBO.

19.2. Optiuni de compilare

Comanda **O** selecteaza urmatorul menu in care se pot face modificari asupra unor valori implicite ale compilatorului. Furnizeaza si o functie help in depistarea erorilor aparute in timpul executiei programului.

Menu-l optiunilor este urmatorul:

```
Compile - Memory
          Com-file
          cHn-file
Command line parameter:
Find run-time error   Quit
```

Memory/Com-file/cHn-file

Cele trei comenzi M, C, H selecteaza modul de compilare, adica alege locul unde va fi depus codul rezultat din compilare.

Memory este modul implicit. Codul este depus in memorie si asteapta sa fie activat printr-o comanda Run.

Tastind C este selectat modul Com-file. Codul rezultat in urma compilarii va fi depus intr-un fisier cu acelasi nume cu fisierul de lucru (sau fisierul principal, daca este specificat), cu extensia .COM. Acest fisier contine codul program si biblioteca cu rutinele Pascal necesare la executia programului si poate fi activat tastindu-i numele. Programele compilate in acest fel pot fi mai mari decit programele compilate in memorie.

Tastind H este selectat modul cHn-file. Codul rezultat in urma compilarii este scris intr-un fisier cu acelasi nume si tipul .CHN. Acest fisier contine codul program, dar nici o biblioteca Pascal si el trebuie activat dintr-un alt program TURBO Pascal cu ajutorul procedurii Chain.

Cind se selecteaza modurile Com sau cHn, menu-ul este expandat cu inca doua linii:

```
Start address : XXXX (min YYYY)
End address   : XXXX (max YYYY).
```

Start address

Adresa de inceput (start address) specifica adresa (in hexazecimal) a primului octet al codului program. In mod normal, aceasta este adresa de sfirsit a bibliotecii Pascal, plus unu, dar poate fi schimbata la o adresa superioara daca se doreste rezervarea unui plus de spatiu (de exemplu, pentru variabile absolute ce vor fi utilizate de programe inlantuite).

Cind se tasteaza **S** sistemul asteapta introducerea unei adrese de inceput. Daca se apasa tasta <RETURN>, atunci adresa de inceput va lua valoarea minima. Nu trebuie data o adresa de inceput mai mica decit valoarea minima, deoarece in acest caz o parte din codul programului se va suprapune peste biblioteca Pascal.

End address

Adresa de sfirsit specifica adresa cea mai mare disponibila programului (in hexazecimal). Valoarea din paranteze indica virful zonei TPA (BDOS-1). Valoarea implicita este cu 700-1000 octeti mai putin decit adresa TPA, pentru a lasa spatiu suficient incarcatorului, rezident putin mai jos de BDOS, atunci cind se executa programe TURBO.

Daca programele compilate vor fi executate pe diverse sisteme de calcul, valoarea adresei de sfirsit poate fi schimbata pentru a corespunde dimensiunii zonei TPA a sistemului respectiv. In acest caz este recomandabil sa se dea valori scazute adresei de sfirsit, de exemplu C100 (48K) sau chiar A100 (40K).

La comanda **E**, sistemul asteapta introducerea adresei de sfirsit. Tastind <RETURN>, adresa de sfirsit va lua valoarea implicita (adresa de inceput a zonei TPA din care s-au scazut 700-1000 octeti). Daca adresa de sfirsit este mai mare decit valoarea implicita, atunci programele rezultate nu pot fi executate din TURBO Pascal, deoarece se vor suprapune peste incarcatorul TURBO.

Parametrii din linia de comanda

Comanda **P** permite introducerea unuia sau mai multor parametri care sint transmisi programului atunci cind acesta este executat in mod Memory. Parametri pot fi accesati prin functiile ParamCount si ParamStr.

Semnalarea erorilor la executie

Daca in timpul executiei unui program compilat in memorie, apare o eroare, este apelat editorul si eroarea respectiva este afisata. Acest lucru nu este posibil daca programul se afla intr-un fisier tip .COM sau tip .CHN. In acest caz va fi tiparit codul erorii si valoarea pe care o avea contorul programului.

Exemplu:

```
Run-time error 01, PC=1B56
Program aborted
```

Pentru a gasi in programul sursa locul unde a aparut eroarea, se poate da comanda **F**, din menu-ul de optiuni, introducindu-se valoarea data de PC.

```
Enter PC : 1B56
```

Linia sursa respectiva va fi afisata, ca si cum eroarea ar fi aparut in timpul executiei unui program compilat in memorie.

19.3. Identificatori standard

Urmatorii identificatori standard sint unici pentru implementarea CP/M-80:

```
Bios        Bdos        ReturPtr
BiosHL      BdosHL      StackPtr
```

19.4. Procedurile Chain si Execute

TURBO Pascal furnizeaza doua proceduri standard, Chain si Execute, care permit activarea altor programe dintr-un program TURBO. Aceste proceduri se apeleaza astfel:

```
Chain (FilVar);
Execute(FilVar);
```

unde FIlVar este o variabila fisier. Daca fisierul exista, este incarcat in memorie si executat.

Procedura Chain este folosita pentru a activa numai fisiere de tip CHN. Un astfel de fisier contine doar codul programului, fara biblioteca Pascal. El este incarcat in memorie si executat la adresa de inceput a programului curent si foloseste biblioteca Pascal prezenta in memorie. Astfel, programul curent si programul inlantuit trebuie sa aiba aceeasi adresa de inceput.

Procedura Execute poate fi folosita pentru a prelucra orice fisier .COM. Fisierul este incarcat si executat la adresa $100, asa cum se specifica in CP/M standard. Daca fisierul disc nu exista apare o eroare I/O. Daca directiva I de compilator este pasiva, executia programului continua cu instructiunea ce urmeaza dupa Chain sau Execute, iar functia IOresult trebuie apelata inainte de urmatoarea intrare/iesire.

Datele pot fi transferate din programul curent in programul inlantuit fie prin variabile globale, fie prin variabile de adresa absoluta. Variabilele globale trebuie sa fie declarate primele, in ambele programe si trebuie sa apara in aceeasi ordine in ambele declaratii. In plus, ambele programe trebuie sa fi fost compilate in sisteme de calcul cu aceeasi dimensiune de memorie. Daca aceste conditii sint satsfacute, variabilele vor fi plasate de catre ambele programe la aceeasi adresa de memorie, si deoarece TURBO Pascal nu initializeaza automat variabilele, ele pot fi distribuite.

Exemplu:

Programul PRINCIPAL.COM:

```
program Principal;
var
 Txt: string[80];
 Cntprg: file;
begin
 write (Introduceti orice text: ');
 readln (Txt);
 assign (Cntprg, 'ContorCar.CHN');
 chain (Cntprg);
end.
```

19.5. Reacoperiri

In mod normal, in timpul executiei, sistemul TURBO presupune ca fisierele de reacoperire se afla pe unitatea de disc selectata implicit. Pentru a schimba valoarea implicita se utilizeaza procedura OvrDrive.

OvrDrive

Sintaxa: OvrDrive (unitate-disc);
in care, unitate-disc poate lua una din valorile: 0=disc implicit, 1=unitatea A,

2=unitatea B etc.

Exemplu:

```
program TestOvr;
overlay procedure ProcA;
begin
.
.
.
end;
overlay procedure ProcB;
begin
.
.
.

end;
procedure Separare;
begin
{procedura  inserata  pentru a separa overlay-urile in  doua
     grupe}
end;
overlay procedure ProcC;
begin
.
.
.
end;
begin
 OvrDrive(2);
 ProcA;
 OvrDrive(0);
 ProcC;
 OvrDrive(2);
 ProcB;
end.
```

19.6. Fisiere

Procedurile Seek si Flush, precum si functiile FilePos si FileSize nu sint aplicabile fisierelor text sub CP/M-80.

19.7. Variabile absolute

Variabilele pot fi declarate astfel incit sa se afle la adrese de memorie specificate, numindu-se in acest caz, variabile absolute. Declaratia unei astfel de variabile se face prin adaugarea cuvintului **absolute** si a unei adrese exprimata printr-o constanta de tip intreg.

Exemplu:

```
var
 IObyte : byte absolute $0003;
 LinieCmd : string [127] absolute $80;
```

O variabila absoluta poate fi declarata la adresa la care se afla o alta variabila absoluta. In acest caz, cuvintul rezervat absolute va fi urmat de identificatorul unei variabile (sau parametru).

Exemplu:

```
var
 Sir :string[32];
 LungSir : byte absolute Sir;
```

De notat ca in declaratia unei variabile absolute este permisa prezenta unui singur identificator. Constructia urmatoare este ilegala:

```
 Id1, Id2 :integer absolute $8000;
```

19.8. Functia Addr

Sintaxa: Addr(Nume);

Functia intoarce in memorie adresa primului octet al tipului, variabilei, procedurii sau functiei cu identificatorul Nume. Daca Nume este o matrice, ea poate fi indexata, iar daca Nume este o inregistrare, cimpurile specifice pot fi selectate. Valoarea returnata este de tip intreg.

19.9. Matrici predefinite

TURBO Pascal ofera doua matrici predefinite de tip byte, numite Mem si Port, care sint folosite pentru accesul direct la memoria CPU si porturi.

Mem

Matricea Mem se utilizeaza pentru acces direct la memorie. Fiecare componenta a matricii este un octet, iar indicii corespund adreselor de memorie. Indexul este de tip intreg. Cind unei componente din Mem i se atribuie o valoare, aceasta este stocata la adresa data de expresia indicelui. Cind matricea Mem este folosita intr-o expresie, se utilizeaza octetul de la adresa specificata de index.

Exemplu:

```
Mem[WSCursor]:=2;
Mem[WSCursor+1]:=$1B;
Mem[WSCursor+2]:=Ord(' ');
IObyte:=Mem[3];
Mem[Addr+Offset]:=Mem[Addr];
```

Matricea Port

Matricea Port se utilizeaza pentru a accesa porturile de date ale CPU Z80. Fiecare element al matricii reprezinta un port, iar indicii corespund numerelor porturilor. Deoarece porturile sint selectate prin adrese pe 8 biti, indicele este de tipul byte. Atunci cind se atribuie o valoare unei componente a matricii Port, aceasta valoare este transmisa portului specificat. Cind intr-o expresie se face o referinta la o componenta a matricii Port, valoarea acesteia este preluata de la portul specificat. Utilizarea matricii Port este restrictionata la folosirea numai in atribuiri si expresii (elementele matricii Port nu pot fi utilizate ca parametri-variabila pentru proceduri sau functii). In plus, operatiile referitoare la intreaga matrice (referiri fara indecsi) nu sint permise.

Optimizarea indexarii matricilor

Directiva X a compilatorului permite programatorului sa aleaga daca indexarea unei matrici va optimiza viteza de executie a programului, sau va

economisi cod program. Modul implicit este cel activ, adica {$X+}, ceea ce produce optimizarea vitezei. Cind directiva este pasiva, adica {$X-}, este minimizata dimensiunea codului.

19.10. Instructiunea With

Numarul implicit al imbricarii instructiunilor With este 2. Directiva de compilator W poate schimba aceasta valoare de la 0 la 9. Pentru fiecare bloc, instructiunile With necesita doi octeti de memorie pentru fiecare nivel de imbricare permis. Pastrind numarul de imbricari la minimum, poate fi influentata zona de date din programele cu multe subprograme.

19.11. Pointeri

MemAvail

Functia standard MemAvail este utila in determinarea spatiului disponibil din heap la un moment dat. Rezultatul este de tip intreg si daca sint disponibili mai mult de 32767 octeti, MemAvail intoarce un numar negativ. Prin urmare, numarul de octeti liberi se calculeaza astfel: 65536.0 + MemAvail.

De notat ca se utilizeaza o constanta reala pentru a genera un rezultat de tip real, deoarece rezultatul este mai mare decit GMaxint.

Pointeri si intregi

Functiile standard Ord si Ptr prevad controlul direct al adreselor continute intr-un pointer. Ord returneaza adresa continuta in argumentul propriului pointer ca un numar de tip intreg, iar Ptr face conversia argumentului sau de tip intreg intr-un pointer care este compatibil cu toate tipurile de pointeri.

Aceste functii sint extrem de pretioase unui programator experimentat, deoarece ele permit ca un pointer sa adreseze orice zona de memorie. Daca insa sint folosite cu neatentie, atunci se pot produce erori grave, deoarece o variabila dinamica poate fi scrisa peste alte variabile sau peste o zona de cod program.

19.12. Apelarea functiilor CP/M

In scopul apelarii rutinelor BDOS si BIOS ale CP/M, TURBO Pascal dispune de doua proceduri standard: Bdos si Bios, si de patru functii standard: Bdos, BdosHL, Bios si BiosHL.

Bdos - procedura si functia

Sintaxa: Bdos(Func{, Param});

Procedura Bdos este folosita pentru apelarea rutinelor CP/M-BDOS. Func si Param sint expresii de tip intreg. Func indica numarul rutinei apelate si este incarcata in registrul C. Param este optional si indica un parametru care este incarcat in registrul pereche DE. Un apel la adresa 5 cheama BDOS.

Functia Bdos este apelata in acelasi mod cu procedura si intoarce un rezultat de tip intreg care reprezinta valoarea returnata de BDOS in registrul A.

Functia BdosHL

Sintaxa: BdosHL(Func{,Param});

Aceasta functie este similara cu functia Bdos de mai sus, exceptind faptul ca rezultatul este valoarea trimisa in registrul pereche HL.

Bios - procedura si functia

Sintaxa: Bios(Func{,Param});

Procedura Bios este folosita pentru apelarea rutinelor BIOS. Func si Param sint expresii der tip intreg. Func indica numarul rutinei apelate, in care 0 desemneaza rutina WBOOT, 1 desemneaza rutina CONST, etc., adica adresa rutinei apelate este Func*3 plus adresa WBOOT continuta in adresele 1 si 2. Param este optional si denumeste un parametru care este incarcat in registrul pereche BC, anterior apelarii.

Functia Bios este apelata asemanator procedurii si intoarce un rezultat intreg care este valoarea returnata de BIOS in registrul A.

Functia BiosHL

Sintaxa: BiosHL(Func,{Param});

Functia este similara cu functia Bios prezentata anterior, cu exceptia faptului ca rezultatul este returnat in registrul HL.

19.13. Drivere I/O definite de utilizator

In unele aplicatii este mai practic pentru programator sa-si defineasca propriile drivere de intrare/iesire. adica rutine care realizeaza I/O de caractere din/in dispozitive externe.

Urmatoarele drivere fac parte din mediul TURBO si sint utilizate de catre driverele I/O standard, desi nu sint disponibile ca proceduri sau functii standard:

```
function   ConSt:boolean;
function   ConIn:char;
procedure  ConOut(Ch:char);
procedure  LstOut(Ch:char);
procedure  AuxOut(Ch:char);
function   AuxIn:char;
procedure  UsrOut(Ch:char);
function   UsrIn:char;
```

Rutina ConSt este apelata de functia KeyPressed.

Rutinele ConIn si ConOut sint utilizate de dispozitivele CON:, TRM: si KBD:.

Rutina LstOut este folosita de dispozitivul LST:, rutinele AuxOut si AuxIn sint utilizate de dispozitivul AUX:, iar rutinele UsrOut si UsrIn sint folosite de dispozitivul USR:.

Aceste drivere utilizeaza implicit punctele de intrare corespunzatoare din BIOS, adica: ConSt utilizeaza CONST, ConIn utilizeaza CONIN, ConOut utilizeaza CONOUT, LstOut utilizeaza LIST, AuxOut utilizeaza PUNCH, AuxIn utilizeaza READER, UsrOut utilizeaza CONOUT si UsrIn utilizeaza CONIN. Totusi, acestea pot fi schimbate de catre utilizator asignind adresa unei proceduri sau functii de tip driver, autodefinite, uneia din urmatoarele variabile standard:

Variabila	Contine adresa
ConStrPtr	functiei ConSt
ConInPtr	functiei ConIn
ConOutPtr	procedurii ConOut
LstOutPtr	procedurii LstOut
AuxOutPtr	procedurii AuxOut

AuxInPtr	functiei AuxIn
UsrOutPtr	procedurii UsrOut
UsrInPtr	functiei UsrIn

O functie sau o procedura driver definita de utilizator trebuie sa corespunda definitiilor date mai sus, adica un driver ConSt trebuie sa fie o functie de tip boolean, un driver ConIn trebuie sa fie o functie de tip char, etc.

19.14. Subprograme externe

Pentru a declara proceduri si functii externe (in general, proceduri si functii scrise in cod masina) se foloseste cuvintul rezervat **external.**

Un subprogram extern nu are nici un bloc, adica nu contine o zona de declaratii si una de instructiuni. Se specifica doar numele subprogramului, urmat imediat de cuvintul external si de o constanta intreaga care defineste adresa din memorie a subprogramului:

```
procedure DiskReset; external $EC00;
function IOstatus:boolean; external $D123;
```

Subprogramele externe pot avea si parametri, sintaxa lor fiind identica cu cea a parametrilor din procedurile si functiile obisnuite:

```
procedure Lot(X, Y:integer); external $F003;
function SortRapid (var List:NrParti); external $1C00;
```

19.15. Inserare de instructiuni in cod masina

TURBO Pascal dispune de instructiunea **inline** ca un mod foarte convenabil de inserare a unor instructiuni in cod masina direct in textul programului. O instructiune inline este formata din cuvintul rezervat inline urmat de unul sau mai multe elemente de cod separate prin "/" si inchise intre paranteze. Un element de cod este construit din unul sau mai multe elemente de date, separate prin semnele plus (+), sau minus (-). Un element de date este fie o constanta intreaga, un identificator de variabila, procedura, functie, fie o referinta la un contor de locatii. O referinta la un contor de locatii este scrisa cu un asterisc (*).

Exemplu:

```
inline(10/$2345/count+1/sort-*+2);
```

Fiecare element genereaza un octet sau un cuvint de cod. Valoarea octetului sau cuvintului este calculata prin adunarea sau scaderea valorilor elementelor de date, corespunzator semnului care le separa. Valoarea unui identificator de variabila este adresa (sau deplasamentul) variabilei. Valoarea unui identificator de procedura sau functie este adresa (sau deplasamentul) procedurii sau functiei. Valoarea unei referinte la o locatie este adresa (sau deplasamentul) contorului de locatii, adica adresa la care se genereaza urmatorul octet de cod.

Un element de cod va genera un octet de cod daca se compune numai dintr-o constanta intreaga cu valoarea cuprinsa intre 0-255. Daca valoarea depaseste 255, sau daca elementul de cod se refera la un identificator de procedura, functie, variabila, sau este o referinta la contorul de locatii, atunci se va genera un cuvint (doi octeti) de cod, cu primul octet cel mai putin semnificativ.

Caracterele "<" si ">" pot fi folosite pentru inhibarea dimensiunii automate descrise mai sus. Daca un element de cod incepe cu "<" va fi

transformat in cod numai octetul cel mai putin semnificativ (chiar daca este o valoare pe 16 biti). Daca un element de cod incepe cu ">", atunci se codifica intotdeauna un cuvint, chiar daca cel mai semnificativ octet este zero.

Exemplu:

```
inline (<$1234/>$44);
```

Instructiunea inline genereaza trei octeti de cod: $34, $44, $00.

Instructiunea inline poate fi plasata oriunde in zona de instructiuni a unui bloc si ea poate utiliza toate registrele CPU. De notat ca registrul SP (stack pointer) trebuie sa aiba acelasi continut la iesire si la intrare.

19.16. Tratarea intreruperilor

Atit sistemul TURBO Pascal, cit si codul generat in urma compilarii, pot fi intrerupte in timpul executiei. Rutinele de intrerupere trebuie sa conserve toate registrele utilizate.

Procedurile cu rutinele de intrerupere pot fi scrise in Pascal. Astfel de proceduri trebuie compilate intotdeauna cu directiva de compilator A activa ({$A+}), nu trebuie sa aiba parametri si trebuie sa asigure ele insele conservarea continutului registrelor care au fost utilizate. Acest lucru se realizeaza plasind o instructiune inline cu instructiunile PUSH respective, chiar la inceputul procedurii, si o instructiune inline cu instructiunile POP, la sfirsitul procedurii. Ultimul element al instructiunii inline finale trebuie sa fie o instructiune EI ($FB), pentru a permite intreruperile ulterioare. Daca sint utilizate intreruperi inlantuite, atunci instructiunea inline trebuie de asemenea, sa specifice o instructiune RETI ($ED, $4D), care sa inhibe instructiunea RET generata de compilator.

Regulile de utilizare a registrelor sint urmatoarele: operatiile cu intregi trebuie sa foloseasca doar registrele AF, BC, DE, HL; celelalte operatii pot utiliza IX si IY, iar operatiile reale vor utiliza registre alternative.

O procedura de intretinere a intreruperilor nu trebuie sa utilizeze operatii I/O care folosesc proceduri si functii standard ale TURBO Pascal, deoarece aceste rutine nu sint reentrante.

In timpul executiei unui program, utilizatorul poate sa permita/inhiba intreruperile prin utilizarea instructiunilor EI si DI generate in instructiunea inline.

Daca se utilizeaza intreruperi de nivel 0 (IM 0) sau 1 (IM 1), programatorul trebuie sa initializeze locatiile de restart (RST 0 nu poate fi folosit, iar locatiile 0-7 sint utilizate de CP/M).

Daca sint folosite intreruperi de nivel 2 (IM 2), programatorul trebuie sa genereze si sa initializeze o tabela de salturi (o matrice de intregi) la o adresa absoluta si sa initializeze registrul I printr-o instructiune inline plasata la inceputul programului.

19.17. Formatul intern al datelor

In continuare, simbolul @ desemneaza adresa primului octet ocupat de o variabila de un anumit tip. Pentru a obtine aceasta valoare se foloseste functia standard Addr.

<u>Tipuri de date de baza</u>

Tipurile de date de baza pot fi grupate in structuri (tablouri, inregistrari, fisiere), dar acest lucru nu afecteaza formatele lor interne.

Tipul scalar

Urmatorii scalari sint memorati fiecare, pe un singur octet: subdomenii ale tipului intreg cu limitele 0-255, boolean, chiar si scalarii declarati cu mai putin de 256 valori posibile. Acest octet contine valoarea ordinala a variabilei.

Urmatorii scalari sint memorati fiecare, pe doi octeti: intregii, subdomenii de tip intreg care nu se incadreaza in limitele 0-256, scalarii declarati cu mai mult de 256 valori posibile. Acesti octeti contin complementul fata de 2 a valorii pe 16 biti, primul octet fiind cel mai putin semnificativ.

Tipul real

Numerele reale ocupa sase octeti de memorie, dind o valoare in virgula mobila, cu mantisa pe 40 biti si exponentul lui 2 pe 8 biti. Exponentul este memorat in primul octet si mantisa in urmatorii cinci octeti, dintre care primul este cel mai putin semnificativ.

```
@       exponent
@+1     octetul cel mai putin semnificativ al mantisei
@+5     octetul cel mai semnificativ al mantisei.
```

Exponentul foloseste un format binar cu deplasamentul $80. Prin urmare, un exponent $84 arata ca valoarea mantisei trebuie multiplicata cu 2^($84-$80) = 2^4 = 16.Daca exponentul este zero, valoarea in virgula mobila este considerata zero.

Valoarea mantisei se obtine prin impartirea intregului fara semn pe 40 biti la 2^40. Mantisa este intotdeauna normalizata (cel mai semnificativ octet, bitul 7 al octetului 5, trebuie considerat 1. Semnul mantisei este stocat in acest bit (1 arata ca numarul este negativ, 0 arata ca numarul este pozitiv).

Tipul string

Un string ocupa un numar de octeti egal cu lungimea maxima a sirului plus 1. Primul octet contine lungimea curenta a sirului. Urmatorii octeti contin caracterele respective, cu primul caracter stocat la adresa cea mai mica. In exemplul de mai jos, L indica lungimea curenta a sirului, iar M indica lungimea maxima:

```
@        lungimea curenta (L)
@+1      primul caracter
@+2      al doilea caracter
.
.
.
@+L      ultimul caracter
@+L+1    neutilizat
.
.
.
@+M      neutilizat.
```

Tipul multime

Un element dintr-o multime ocupa un bit si deoarece numarul maxim de elemente dintr-o multime este 256, o variabila de tip multime nu va ocupa niciodata mai mult de 32 octeti.

Daca o multime contine mai putin de 256 elemente, unii biti sint fortati pe zero si deci nu este nevoie sa fie stocati. Pentru o utilizare eficienta a memoriei, cea mai buna metoda de a stoca o variabila multime de un tip dat este ignorarea tuturor bitilor nesemnificativi si rotirea bitilor care au ramas,

astfel incit primul element al multimii sa ocupe primul bit al primului octet. Operatiile de rotire sint insa, lente si de aceea TURBO recurge la un compromis: sint stocati numai octetii care nu sint static zero (sint memorati octetii care au cel putin un bit utilizat). Aceasta metoda de comprimare este foarte rapida si in cele mai multe cazuri este la fel de eficienta in utilizarea memoriei ca metoda rotirii.

Numarul de octeti ocupati de o astfel de variabila se calculeaza astfel:

(Max div 8) - (Min div 8) + 1

in care, Max si Min sint limitele superioara si inferioara ale tipului de baza al multimii. Adresa din memorie a unui element specific E este:

AdresaMem = @ + (E div 8) - (Min div 8)

si adresa bitului din octetul de la AdresaMem este:

AdresaBit = E mod 8

in care E indica valoarea ordinala a elementului.

Blocuri de interfata cu fisiere

Tabela urmatoare arata formatul unui bloc de interfata fisier din TURBO Pascal:

@+0	octet flag
@+1	buffer-ul pentru caractere
@+2	pointer la buffer-ul de sector(LSB)
@+3	pointer la bufferul de sector (MSB)
@+4	numarul de inregistrari (LSB)
@+5	numarul de inregistrari (MSB)
@+6	lungimea inregistrarii (LSB)
@+7	lungimea inregistrarii (MSB)
@+8	inregistrarea curenta (LSB)
@+9	inregistrarea curenta (MSB)
@+10	neutilizat
@+11	neutilizat
@+12	primul octet din CP/M FCB
.	
.	
.	
@+47	ultimul octet din CP/M FCB
@+48	primul octet al buffer-ului de sector
.	
.	
.	
@+175	ultimul octet al buffer-ului de sector.

Formatul ooctetului flag de la @+0 este urmatorul:

bit 0..3	tip fisier
bit 4	semafor read
bit 5	semafor write
bit 6	marcator output
bit 7	marcator input.

Tipul fisierului poate fi 0 pentru fisiere pe disc si 1-5 pentru dispozitivele logice de intrare. Pentru fisierele fara tip, bitul 4 este setat daca continutul buffer-ului de sector este nedefinit, iar bitul 5 este setat daca s-a scris in buffer-ul de sector. Pentru fisiere text, bitul 5 este setat daca buffer-ul de caractere contine un caracter citit anterior. Bitul 6 este setat daca este permisa iesirea, bitul 7 este setat daca este permisa intrarea.

Pointer-ul buffer-ului de sector stocheaza un deplasament (0..127) in buffer-ul de sector la @+48. Pentru fisiere cu sau fara tip, cele trei cuvinte de la @+4 la @+9 memoreaza numarul de inregistrari din fisier, lungimea inregistrarii in octeti si numarul inregistrarii curente. Intr-un fisier fara

tip, FIB nu are buffer de sector si deci pointer-ul de buffer de sector nu este utilizat. Cind un fisier text este asignat al un dispozitiv logic, sint utilizati numai octetul flag si buffer-ul de caractere.

Tipul pointer

Un pointer este format din doi octeti care contin o adresa de memorie pe 16 biti, si este memorat cu octetul cel mai putin semnificativ in stinga.

Tipuri de date structurate

Tablouri

Componentele cu valorile cele mai mici ale indicelui sint stocate la adresele cele mai mici. O matrice multidimensionala este stocata cu dimensiunea cea mai din dreapta crescind prima.

Inregistrari

Primul cimp al unei inregistrari este stocat la adresa cea mai mica. Daca inregistrarea nu contine parti variante, lungimea este data de suma lungimilor cimpurilor individuale. Daca o inregistrare are variante, numarul total de octeti ocupati de inregistrare este dat de lungimea partii fixe plus lungimea celei mai mari parti variabile. Fiecare varianta incepe la aceeasi adresa de memorie.

Fisiere disc

Fisierele disc difera de alte structuri de date prin aceea ca datele nu sint memorate in memoria interna, ci intr-un fisier pe un dispozitiv extern. Un fisier disc este controlat prin intermediul unui bloc de interfata cu fisierul (FIB), asa cum a fost descris mai sus. Exista doua tipuri de fisiere disc: fisiere in acces direct si fisiere text.

Fisiere in acces direct

Un fisier cu acces direct este format dintr-un numar de inregistrari, cu aceeasi lungime si acelasi format intern. Pentru optimizarea spatiului de stocare, inregistrarile unui fisier sint contigue. Primii patru octeti ai primului sector al unui fisier contin numarul de inregistrari si lungimea in octeti a fiecarei inregistrari. Prima inregistrare a fisierului este stocata incepind cu al patrulea octet.

sector 0, octet 0	numarul de inregistrari (LSB)
sector 0, octet 1	numarul de inregistrari (MSB)
sector 0, octet 2	lungimea inregistrarii (LSB)
sector 0, octet 3	lungimea inregistrarii (MSB).

Fisiere text

Componentele de baza ale unui fisier text sint caracterele, dar un fisier text este divizat in linii. Fiecare linie este formata dintr-un numar oarecare de caractere terminat printr-o secventa CR/LF (cod ASCII $0D/$0A). Fisierul este terminat prin CTRL/Z (cod ASCII $1A).

Parametri

Parametrii sint transferati procedurilor si functiilor prin intermediul stivei Z-80. In mod normal, acest lucru nu prezinta interes pentru programator, deoarece codul masina generat de TURBO Pascal va scrie (PUSH) automat parametri in stiva inaintea unei apelari si ii va extrage (POP) plasindu-i la inceputul subprogramului. Totusi, daca programatorul doreste sa utilizeze subprograme

externe, acestea trebuie sa plaseze ele insele parametri in stiva.

La intrarea intr-un subprogram extern, virful stivei contine intotdeauna adresa de intoarcere (un cuvint). Parametri, daca exista, sint plasati linga adresa de intoarcere, adica la adresele cele mai mari ale stivei. Deci, pentru a avea acces la parametri, subrutina trebuie sa extraga (POP) mai intii adresa de intoarcere, apoi parametri, dupa care trebuie sa puna la loc in stiva adresa de intorcere (PUSH).

Parametri-variabila

Cu un parametru-variabila se transfera un cuvint in stiva, care da adresa absoluta de memorie a primului octet ocupat de parametrul actual.

Parametri valoare

Cu parametri valoare, datele sint transmise in stiva, in functie de tipul parametrului, dupa cum este descris mai jos.

Scalari

Scalarii de tip intreg, boolean, char si scalarii declarati sint transferati in stiva pe un cuvint. Daca variabila ocupa un singur octet atunci cind este stocata, atunci octetul cel mai semnificativ este zero. Un cuvint este extras din stiva utilizind instructiunea POP HL.

Reali

Un parametru de tip real este transferat in stiva utilizind sase octeti. Daca acesti octeti sint extrasi utilizind secventa de instructiuni:

```
POP  HL
POP  DE
POP  BC
```

atunci, L va contine exponentul, H va contine al cincilea octet al mantisei (cel mai putin semnificativ), E va contine al 4-lea, D va contine al 3-lea, C al 2-lea si B primul (cel mai semnificativ).

Siruri

Cind un sir se afla in virful stivei, octetul indicat de SP contine lungimea sirului. Octetii de la adresele SP+1 pina la SP+n, unde n este lungimea sirului, contin sirul. Primul caracter este stocat la adresa cea mai mica.

Multimi

O multime ocupa intotdeauna 32 de octeti in stiva (comprimarea se utilizeaza numai la incarcarea si stocarea multimilor).

Pointeri

Valoarea unui pointer este transferata in stiva pe un cuvint care contine adresa de memorie a unei variabile dinamice. Valoarea nil corespunde unui cuvint zero.

Matrici si inregistrari

Chiar si atunci cind sint utilizate ca parametri valoare, parametri de tip valoare sau inregistrare nu sint plasati in stiva. In schimb este stocat un

cuvint care contine adresa primului octet al parametrului. Subrutina scrisa de programator este responsabila cu extragerea acestui cuvint din stiva si utilizarea lui ca adresa sursa intr-o operatie de copiere bloc.

19.18. Rezultatele functiilor

Functiile externe scrise de utilizator trebuie sa intoarca rezultatul astfel:

- valorile tipurilor scalare trebuie intoarse in registrul pereche HL. Daca tipul rezultatului este exprimat pe un octet, atunci registrul L va contine rezultatul, iar registrul H trebuie sa fie zero;
- rezultatele reale trebuie intoarse in registrele pereche BC, DE, HL. Registrele B, C, D, E, H trebuie sa contina mantisa, iar L trebuie sa contina exponentul;
- sirurile si multimile trebuie intoarse in virful stivei in formatele descrise mai sus;
- valorile pointerilor trebuie returnate in registrul pereche HL.

19.19. Structuri de memorie

Heap si Stack

Dupa cum indica imaginea memoriei prezentata in sectiunile precedente, in timpul executiei unui program sint mentinute trei structuri asemanatoare stivei (stack): heap, stiva CPU, stiva recursiva.

Heap este folosita pentru a memora variabilele dinamice si este controlata de procedurile standard New, Mark si Release. La inceputul unui program, pointerul de heap (HeapPtr) este setat la adresa cea mai de jos a zonei libere de memorie (primii trei octeti dupa codul obiect).

Stiva CPU este utilizata pentru a memora rezultate intermediare in timpul evaluarii expresiilor si pentru a transmite parametri la proceduri si functii. De asemenea, stiva CPU este folosita de instructiunea **for** atunci cind este activa, ocupind un cuvint. La inceputul unui program, pointer-ul stivei CPU este setat la adresa ce indica virful zonei libere din memorie.

Stiva recursiva este utilizata doar de catre procedurile si functiile recursive. La intrarea intr-un subprogram recursiv, acesta isi copiaza spatiul sau de lucru in stiva recursiva, iar la iesire intregul spatiu de lucru este restaurat la starea initiala. Valoarea initiala implicita a pointer-ului RecurPtr, la inceputul unui program, este 1K ($400) sub pointer-ul stivei CPU. Datorita acestei tehnici, variabilele locale intr-un program nu pot fi utilizate ca parametri-variabila in apelarile recursive.

Variabilele predefinite: HeapPtr, RecurStr, StackPtr permit programatorului sa controleze pozitia heap-ului si a stivei. Tipul acestor variabile este integer. De notat ca HeapPtr si RecurPtr pot fi utilizati in acealasi context ca orice variabila, in timp ce StackPtr poate fi folosita doar in atribuiri si expresii. Cind se lucreaza cu aceste variabile, trebuie avuta grija intotdeauna ca ele sa trimita la adrese din cadrul zonei libere de memorie si ca:

HeapPtr < RecurPtr < StackPtr.

Nerespectarea acestor reguli produce rezultate imprevizibile, probabil fatale.

19.20. Administrarea memoriei

Compilarea in memorie

In timpul compilarii unui program in memorie (mod de compilare M), memoria

arata astfel:

```
 _____________________      0000
|_________________|__       CP/M si spatiu de lucru la executie
|_________________|__       Biblioteca Pascal
|_________________|__       Interfata TURBO, editor si compilator
|_ _ _ _ _ _ _ _ _| _       Mesaje de eroare, optional
|                 |
|                 |         Text sursa
|_ _ _ _ _ _ _ _ _| _
|                 | |       Cod obiect crescind in sus
|                 | v
|                 | ^
|                 | |       Tabela de simboluri crescind in jos
|_________________|__
|                 | ^
|_________________| |       Stiva CPU crescind in jos
|                 |
|                 |         CP/M
|_________________|__       HighMem
```

Daca fisierul cu mesajele de eroare nu este incarcat cind intra in executie TURBO, atunci textul sursa incepe mult mai jos in memorie.

Compilarea unui fisier pe disc

In timpul compilarii unui fisier .COM sau .CHN memoria arata asemanator cu cea din timpul compilarii in memorie, exceptind codul obiect generat, care nu se gaseste in memorie, ci este scris intr-un fisier pe disc. De asemenea, codul incepe la adrese mai inalte (chiar dupa biblioteca Pascal, in loc de a fi dupa textul sursa), ceea ce face posibila compilarea unor programe de dimensiuni mai mari.

Executia in memorie

Dupa compilarea unui program, locul unde se afla in memorie sfirsitul codului obiect este cunoscut. Pointer-ul de heap este pozitionat implicit pe aceasta valoare si heap-ul creste in sus catre stiva recursiva. Dimensiunea maxima a memoriei este BDOS-1 (indicata in menu-ul de optiuni de compilare). Variabilele programului sint stocate incepind de la aceasta adresa in jos. Sfirsitul zonei ocupata de variabile este "virful zonei libere de memorie", care este valoarea initiala a pointer-ului stivei CPU.

Executia unui fisier program

Un program poate fi executat fie printr-o comanda Run din modul de compilare direct, selectat din menu-ul de optiuni, fie printr-o comanda eXecute, fie direct din CP/M. Adresa implicita de start a programului este primul octet liber dupa biblioteca Pascal utilizata in executie. Aceasta valoare poate fi manipulata impreuna cu adresa de start din menu-ul de compilare pentru a crea spatiu pentru variabilele absolute sau procedurile externe, intre biblioteca Pascal si cod. Dimensiunea maxima a memoriei este BDOS-1, iar valoarea implicita este determinata de locatia BDOS in sistemul de calcul utilizat.

Intreprinderea de Echipamente Periferice Bucuresti

CP/M

COMPILATORUL LIMBAJULUI "C"

1986

CUPRINS

Pagina

Cap.I Introducere 4

Cap.II Utilizare compilatorului de C SuperSoft 6
- fisierele de lucru sub CP/M 6
- fisierele necesare 7
- formatul liniilor de comanda 7
- compilarea sub ASM80 8
- compilarea sub ASM86 9
- compilarea sub M80 si L80 10
- optiunile liniei de comanda CC 11
- optiunile liniei de comanda CC dependente de masina 11
- optiunile liniei de comanda C2 12
- directivele preprocesorului 13
#DEFINE 13
#INCLUDE 14
#ASM , #ENDASM 14
- relocarea codului generat de compilator 15
- declararea functiilor de listare 15

Cap.III Functiile bibliotecii standard de C SuperSoft 17
- incorporarea functiilor bibliotecii standard in program 17
- initializarea functiilor de alocare dinamica a memoriei 19
- descrierea functiilor bibliotecii standard 19
ABS 20
ALLOC
ATOI
BDOS 21
BIOS
BRK
CCAI 22
CCALA
CLOSE
CODEND 23
CPMVER
CREAT
ENDEXT
EVNBRK 24
EXEC
EXECL
EXIT 25
EXTERNS
FABORT
FCLOSE
FFLUSH 26
FGETS
FOPEN 27
FPRINTF
FPUTS 28
FREE
FSCANF 29
GETC

GETCHAR 29
GETS
GETVAL 30
GETW
INDEX
INITB 31
INITW
INP
ISALNUM 32
ISALPHA
ISASCII
ISCNTRL
ISDIGIT
ISLOWER
ISNUMERIC 33
ISPRINT1
ISPUNCT
ISSPACE
ISUPPER
ISWHITE 34
KBHIT
MOVMEM
OPEN
OTELL 35
OUTP
PAUSE
PEEK
PGETC
POKE 36
PPUTC
PRINTF
PUTC 37
PUTCHAR
PUTS 38
PUTW
RAND
READ 39
RENAME
RESET
RINDEX 40
RTELL
SBRK
SCANF 41
SEEK 42
SETEXIT 43
SETMEM
SLEEP
SPRINTF
SRAND
SSCANF 44
STRCAT
STRCMP
STRCPY
STREQ 45
STRLEN
STRNCAT
STRNCPY
TELL
TOLOWER 46
TOPOFMEM

TOUPPER 46
UNGETC
UGETCHAR
UNLINK 47
WRITE

Cap.IV Inserarea rutinelor proprii in biblioteca de executie C2.RH 48

Anexa A Diferentele dintre C SuperSoft si C Standard 51
Anexa B Configuratii disponibile...52
Anexa C Unele probleme uzuale si solutii 53
Anexa D Locatiile functiilor furnizate 54

CAPITOLUL I

Introducere

Compilatorul de C SuperSoft accepta intregul limbaj C, cu citeva exceptii care sint prezentate amanuntit in Anexa A a acestui manual.

"Limbajul de programare C" scris de Brian W. Kernighan si Dennis M. Ritchie (Englewood Cliffs NJ) Prentice - Hall Inc., 1978 este referinta standard a limbajului C necesara oricarui programator in acest limbaj.

Un articol de mare interes, care prezinta evolutia si principiile generale ale limbajului C este "Limbajul de programare C" scris de D. M. Ritchie si altii in The Bell System Technical Journal, 57(6), Iulie - August 1978, pg. 1991-2019.

Compilatorul de C SuperSoft e un compilator in 2 pasi, de optimizare si auto-control care genereaza un fisier final de iesire in cod sursa de asamblare. Compilatorul e adaptabil unei game largi de sisteme de operare si CPU-uri. Configuratiile sistemului de operare si masinilor disponibile in mod curent sint prezentate in Anexa B.

Datorita portabilitatii inerente limbajului C si a implementarii specifice in compilatorul Dv., configuratiile pentru alte sisteme de operare si masini pot fi realizate usor si rapid. Serviciile de "porting" sint disponibile de la SuperSoft.

Scrierea compilatorul in limbajul pe care-l implementeaza a dus nu numai la dezvoltarea si optimizarea lui, dar a si constituit un mijloc direct de testare a compilatorului. Acesta a fost apoi supus unor testari riguroase. In prezent, limbajul C SuperSoft este folosit la cele mai multe proiecte de programare.

Ca rezultat al optimizarilor pe care le face, codul generat de compilator are o eficienta spatiala si temporala adecvata atit programarii de sisteme de operare, cit si de aplicatii. Un exemplu al acestei eficiente consta in faptul ca pentru cel putin un sistem de operare, CP/M, marca inregistrata a Digital Research Inc., intreaga interfata I/E a compilatorului e scrisa in limbaj C SuperSoft, cu exceptia a 10 linii scrise in limbaj de asamblare, necesare legarii cu sistemul de operare.

Compilatorul de C SuperSoft e un compilator modular in 2 pasi. Fiecare pas e realizat de o secventa separata si distincta a compilatorului, oferind astfel o serie de avantaje importante. Divizarea compilatorului in 2 programe de sine statatoare ii permite sa accepte aproape tot limbajul C, ocupind totusi o memorie relativ mica.

Aceasta structura modulara determina existenta unei interfete naturale intre primul pas: CC - analizorul (Parser) si al doilea pas - optimizorul/generatorul (C2). Fisierul de iesire al primului pas devine fisierul de intrare al celui de al doilea pas. Modularitatea faciliteaza si posibilitatea de adaptare a compilatorului la alte masini si sisteme de operare, deoarece primul modul e independent de sistem si de masina, si trebuie schimbate doar portiuni din cel de-al doilea modul.

Primul pas (CC) al compilatorului accepta ca intrare un fisier cod sursa de C SuperSoft si il analizeaza. Ca iesire el genereaza un fisier intr-un cod intermediar: cod-Universal, sau cod-U (un cod implicit care nu e specific nici unui sistem sau masina). Deoarece una din specificatiile de proiectare ale acestui compilator a fost aceea ca iesirile ambilor pasi sa fie inteligibile pentru om, codul-U poate fi revizuit si modificat folosind un editor obisnuit de text. Cu toate ca nu este inca disponibil, este in studiu un optimizor independent de masina din cod-U in cod-U.

Al doilea pas (C2) al compilatorului accepta ca intrare fisierul in cod-U generat de CC. Fisierul de intrare e supus unui proces complex de optimizare implicind rearanjarea si revizuirea codului in cadrul functiunilor, precum si transformarile de cod local reiterative (printr-o lupa la unui anumit numar de linii de cod). Un proces de translare e integrat cu acest proces de optimizare pentru a genera un fisier final de iesire in codul sursa de asamblare.

Din alegerea codului sursa de asamblare ca iesire finala a compilatorului rezulta citeva avantaje, unele fiind de valoare deosebita pentru programatorul la nivel de sistem. Deoarece codul generat e intentionat nespecific nici unui asamblor folosit pe o anumita masina, practic poate fi folosit orice asamblor compatibil hard (inclusiv unul care produce cod relocabil). Astfel iesirea acstui compilator poate sa fie integrata in orice sistem soft in care codul sursa a fost scris in (sau e compatibil cu) limbaj de asamblare.

Programatorul poate sa insereze linii de cod de asamblare direct intr-un fisier sursa C prin directivele compilatorului #ASM si #ENDASM. Liniile astfel inserate nu sint optimizate sau alterate in nici un fel de catre compilator.

Utilizarea codului sursa de asamblare ca limbaj de iesire din compilator, satisface specificatiile de proiectare care cer ca iesirile ambilor pasi sa fie inteligibile omului. Deoarece codul generat nu e intotdeauna cel dorit, programatorul poate sa modifice codul sursa sau sa "curete" codul generat pentru ca rezultatul sa fie mai usor adaptabil unei anumite aplicatii.

Un mare neajuns pentru compilator e timpul necesar pentru a compila un program in raport cu timpul necesar pentru a-l executa. Scopul principal in proiectarea compilatorului fiind acela de a genera un cod destul de eficient pentru programarea la nivel de sistem, s-a marit viteza de executie cu pretul vitezei de compilare (unele optimizari realizate de C2 solicita un timp proportional cu patratul celei mai mari functii de otimizat).

Marirea vitezei ofera compilatorului posibilitatea de a fi aplicabil unei game largi de sarcini de programare, sporind insa timpul de dezvoltare a programului. Compilatorul de C este unic in ceea ce priveste posibilitatea programarii eficiente la nivel de sistem, structurata si clara pe un sistem hard relativ mic.

Capitolele urmatoare va ofera informatiile necesare pentru utilizarea acestui compilator, precum si descrierea unor caracteristici.

Capitolul 2 - prezinta instructiunile de utilizare a compilatorului. Sint notate directivele de preprocesare si utilizarea lor.

Capitolul 3 - descrie functiile standard de biblioteca furnizate cu compilatorul.

Capitolul 4 - ilustreaza modul de inserare a codului in biblioteca de executie al lui C2.RH.

Anexele A si B - prezinta diferentele dintre limbajul de C SuperSoft si versiunea a 7-a UNIX C standard, pe de-o parte, si configuratiile curente ale sistemului de operare a compilatorului de C SuperSoft, pe de alta parte, astfel incit sa poata fi mai usor actualizate.

CAPITOLUL II

Utilizarea Compilatorului de C SuperSoft

Subiectele prezentate

Acest capitol prezinta informatiile necesare pentru a utiliza optim compilatorul de C SuperSoft. Informatiile sint prezentate cit mai concis si intr-o succesiune naturala. Subiectele prezenate sint:

- ce fisiere sint furnizate pe discul compilatorului,
- care fisiere va sint necesare pentru a va compila programele,
- ce alta parte soft va mai este necesara,
- sintaxa liniilor de comanda ale compilatorului,
- utilizarea compilatorului intr-un sistem de operare specific (CP/M de exemplu),
- optiunile liniei de comanda disponibile pentru CC si C2,
- directivele preprocesorului compilatorului, si in final,
- relocarea codului final de iesire al compilatorului pentru orice locatie RAM sau ROM.

PENTRU PROTECTIE faceti o copie a discului Dv. cu compilatorul de C Super-Soft imediat ce il primiti. Pastrati originalul intr-un loc sigur, ca disc martor, si pentru toate activitatile Dv. ulterioare folositi doar copia.

Fisierele de lucru sub CP/M

Daca aveti versiunea CP/M pe copia de lucru, veti gasi urmatoarele fisiere:

```
CC.ext     : primul pas al compilatorului, Analizorul (Parser)
C2.ext     : al doilea pas, Optimizorul/Generatorul de cod
C2.RH      : fisierul "header" folosit de C2 pentru ASM
C2.RT      : fisierul "trailer" folosit de C2 pentru ASM
C2.RTM     : "trailer"-ul pentru RMAC si M80
C.SUB      : fisierul CP/M SUBMIT pentru compilare
STDIO.H    : "header"-ul functiilor standard I/E
STDIO.C    : functiile standard I/E
ALLOC.C    : functiile de alocare dinamica a memoriei
CRUNT2.C   : portiuni ale limbajului de C pentru executie
FUNC.C     : functii auxiliare
FORMATIO.C: functii printf, scanf si altele asemanatoare
*.REL      : fisiere necesare compilarii cu RMAC (Digital Research) sau M80
             (Microsoft)
SAMP?.C    : programe exemplu care testeaza diferite caracteristici ale
             limbajului C.
```

CC.ext si C2.ext sint fisierele compilatorului. Extensia (.ext) corecta pentru aceste 2 fisiere de lucru depinde de sistemul Dv. de lucru. Daca lucrati sub CP/M-80 extensia corecta e ".COM". CR.RT, STDIO.H, ALLOC.H si AUXFN.H sint fisiere cu functii incluse. (Celelalte fisiere de pe disc cu extensia ".C" sint programe in codul sursa de C SuperSoft oferite ca exemplu.)

Fisierele ".REL" sint incluse pentru utilizatorii care au pachetul asamblor RMAC-Digital Research sau M80-Microsoft. Daca nu aveti un asamblor compatibil cu acestea, nu luati in considerare aceste fisiere.

In cazul compilatorului 8086, al doilea pas e divizat in COD2COD (Optimizorul) si C2I86 (Generatorul de cod). C2.RH si C2.RT sint inlocuiti de

C2I86.RH si C2I86.RT. Fisierele ".REL" si C2.RTM nu sint incluse.

Fisierele necesare

Pentru a compila un program, intotdeauna va sint necesare primele 4 fisiere mentionate mai sus, daca nu folositi un asamblor relocabil. Cei care folosesc un asamblor relocabil vor avea nevoie de C2RT.REL (vezi Anexa E, pentru informatii referitoare la modul de creare al fisierului C2RT.MAC).

Al doilea pas al compilatorului, C2, incorporeaza automat in programul Dv. toate segmentele de cod si functiile cuprinse in biblioteca de executie si C2.RH, C2.RT in timpul compilarii daca nu e folosita relocarea. Altfel, utilizatorul va trebui sa link-editeze functiile bibliotecii de executie.

Fisierele ".H" sau ".C" vor fi necesare doar daca programul va solicita una sau mai multe functii. Oricum, aceste functii stau la baza formarii programelor, astfel incit multe dintre ele vor fi apelate in majoritatea aplicatiilor. Daca programul Dv. va apela oricare dintre functiile definite in ultimele 3 fisiere, trebuie sa urmariti ca programul sa contina toate definitiile necesare de functii si date din aceste fisiere (inclusiv cele apelate sau solicitate de functiile apelate de programul Dv.).

La inceputul capitolului 3 sint prezentate 4 metode de realizare a apelarii. Anexa D prezinta un rezumat al functiilor de C prevazute in fisierele respective.

Deoarece iesirea finala a acestui compilator e in cod sursa de asamblare a masinii Dv., aveti nevoie de un asamblor compatibil cu hard-ul si sistemul de operare plus soft-ul necesar pentru a incarca si rula programele in sistem.

Formatul liniilor de comanda

Pentru a compila un program sursa de C SuperSoft folositi urmatoarele fisiere de comanda in secventa:

```
CC filename.C ... option ...
C2 filemane.COD option ...
```

Cind invocati CC, puteti introduce oricite nume de fisier in linia de comanda, separate prin spatiu (blank). Filename se refera la partea prefix a specificatiei fisier necesara pentru sistemul de operare. Nu este necesara explicitarea functiilor in fiecare din fisierele specificate in linia de comanda, deoarece toate vor fi analizate in ordinea mentionata, ca si cum ar fi un singur fisier. Fisierul de iesire in cod-U va avea acelasi nume ca primul fisier dat. Cind executati C2 puteti specifica doar un nume fisier.

Prin conventie, fisierele în cod sursa C au extensia sau sufixul ".C", cu toate ca orice extensie e permisa. CC asigura automat extensia ".COD" pentru fisierele sale de iesire in cod-U. Fisierelor de iesire in cod de asamblare ale lui C2 li se atribuie automat extensia ".ASM".

Optiunile liniei de comanda pentru CC si C2 trebuie sa fie separate prin "spatiu". Deoarece conditiile implicite specifica de obicei modul dorit de functionare al compilatorului, nici o optiune nu este necesara.

Totusi, daca rulati compilatorul sub un CPU, altul decit un 8080, 8085, 8086 sau Z80 (seria 8080), trebuie sa setati cele 3 optiuni dependente de masina, +J, -I si -P (descrise in pag. 10-11), la valorile lor corecte pentru procesorul Dv.

Ca exemplu al procedurii pe care trebuie sa o aplicati pentru a compila si rula un program de C SuperSoft pe o anumita masina si intr-un anumit sistem de operare, sint descrise procedeele de lucru sub CP/M-80 pentru CPU din seria 8080. Presupunem ca sinteti pregatiti sa va compilati si sa va rulati primul program C, SAMP1.C . Puteti incepe dupa ce mai intii ati verificat ca aveti pe disc toate fisierele necesare si ca programul Dv. contine toate definitiile de functii si date necesare (cu exceptia celor definite in C2.RT, vezi inceputul

capitolului 3).

Compilarea sub ASM, Asamblor standard CP/M-80

Pentru a transforma SAMP1.C intr-un fisier de comanda executabil astfel incit sa poata fi rulat sub CP/M-80, trebuie sa tastati fiecare dintre comenzile de mai jos in secventa (aceasta procedura e similara celei pentru MAC). (Executarea fiecarei dintre aceste comenzi CP/M incepe indata ce apasati pe CR la sfirsitul liniei.) Tastati:

```
CC SAMP1.C
C2 SAMP1.COD +ASM
ASM SAMP1
LOAD SAMP1
```

Fiecare dintre comenzile de mai sus provoaca crearea unui fisier nou pe discul care contine SAMP1.C. Numele acestor fisiere sint:

```
SAMP1.COD      ; iesirea de la CC
SAMP1.ASM      ; iesirea de la C2
SAMP1.HEX      ; iesirea de la ASM
SAMP1.COM      ; iesirea de la LOAD
```

Ultimul fisier creat e fisierul de comanda executabil pentru programul Dv. Aceasta e singura forma in care poate fi rulat programul sub CP/M. (Celelalte fisiere pot fi sterse.) Dupa ce ati creat acest fisier, tastati doar:

```
SAMP1
```

-- apasati pe CR si va incepe executia programului.

CC, C2 si ASM pot genera si mesaje de eroare indicind defectele din program. Mesajele de eroare generate de CC si C2 vor fi prezentate intr-un capitol special care va constitui actualizarea acestui manual. Documentatia CP/M ofera informatii despre mesajele de eroare generate de ASM. Doua tipuri de mesaje de la ASM isi au originea in programele C insuficient elaborate:

1) Erori de faza si etichete multiplu definite,in general provoacate de redefinirea unui simbol extern C (inclusiv functii C). Atentie, trebuie corelat numarul maxim de caractere semnificative din numele unui simbol extern cu versiunea de asamblor folosit (primele 6 la MAC sau M80 si primele 8 la ASM).

2) Etichetele nedefinite indica de obicei module externe C nedefinite (inclusiv functii C).

In cursul dezvoltarii programelor, sinteti nevoit sa repetati procedura descrisa mai sus, schimbind doar numele fisierului sursa. S-ar economisi timp daca ati putea face ca intreaga procedura sa poata fi executata printr-o singura comanda. Sub CP/M , comanda SUBMIT si fisierele SUBMIT ofera modul de realizare a acestui lucru. Alte sisteme de operare ofera facilitati similare.

Un fisier SUBMIT adecvat, cu un nume fisier C.SUB va contine urmatoarele linii de comanda:

```
CC $1.C
C2 $1.COD +ASM
ASM $1.AAZ
LOAD $1
$1
```

-- unde "$1" este un parametru simbolic care va fi ulterior inlocuit de

primul parametru real al unei comenzi SUBMIT.

Dupa ce ati creat un astfel de fisier pe disc, tastati:

```
SUBMIT C SAMP1
```

-- si apasati pe CR. Daca CP/M gaseste C.SUB pe discul curent, vor fi executate urmatoarele 5 comenzi (SAMP1 a fost substituit pentru fiecare "$1"):

```
CC SAMP1.C
C2 SAMP1.COD +ASM
ASM SAMP1
LOAD SAMP1
SAMP1
```

Pentru informatii suplimentare despre fisierele SUBMIT consultati documentatia CP/M sau unul dintre manualele CP/M disponibile.

Compilarea sub ASM86, Asamblor standard CP/M-86

Pentru a transforma SAMP1.C intr-un fisier de comanda executabil astfel incit sa poata fi rulat sub CP/M-86, trebuie sa tastati in secventa fiecare dintre comenzile prezentate mai jos. Tastati:

```
CC SAMP1.C
COD2COD SAMP1.COD
C2I86 SAMP1.U
ASM86 SAMP1
GENCMD SAMP1 DATA[X1000]
```

Fiecare dintre comenzile de mai sus creaza un fisier nou pe discul care contine SAMP1.C . Numele acestor fisiere sint:

```
SAMP1.COD    ; iesirea din CC
SAMP1.U      ; iesirea din COD2COD
SAMP1.A86    ; iesirea din C2I86
SAMP1.H86    ; iesirea din ASM86
SAMP1.CMD    ; iesirea din GENCMD
```

Ultimul fisier creat este fisierul de comenzi executabile pentru program. Aceasta e singura forma cu care programul poate fi rulat sub CP/M-86. (Celelalte pot fi sterse.) Indata ce acest fisier a fost creat, trebuie sa tastati doar:

```
SAMP1
```

-- apasati pe CR si va incepe executia programului.

CC, COD2COD, C2I86 si ASM86 pot genera si mesaje de eroare indicind defectele din program. Mesajele de eroare generate de CC si C2, in mare masura auto-explicative, vor fi prezentate intr-un capitol prevazut ca actualizare a acestui manual. Documentatia CP/M-86 ofera informatii despre mesajele de eroare generate de ASM86. Doua tipuri de mesaje generate de ASM86 pot fi datorate programelor in C insuficient elaborate:

1) Erorile de faza si etichetele multiplu definite sint adesea create de redefinirea modulelor externe C (inclusiv functii C). Atentie la numarul de caractere semnificative din numele modulelor externe - trebuie sa fie corelat cu cel acceptat de asamblorul (si incarcatorul) folosit.

2) Etichetele nedefinite indica de obicei module externe C nedefiniti (inclusiv functii C).

Pe parcursul dezvoltarii programelor, sinteti nevoit sa repetati procedura

descrisa, schimbind doar numele fisierului de intrare. S-ar putea economisi timp daca intreaga procedura ar putea fi realizata printr-o singura comanda. Sub CP/M, comanda si fisierele SUBMIT ofera posibilitatea realizarii acestui lucru.

Un fisier SUBMIT corect, cu nume fisier C.SUB, va contine urmatoarele linii de comanda:

```
CC $1.C
COD2COD $1.COD
C2I86 $1.U
ASM86 $1 $$PZ
GENCMD $1 DATA[X1000]
```

-- unde "$1" este un parametru simbolic care va fi schimbat ulterior cu primul parametru real al unei comenzi SUBMIT.

Dupa ce ati creat un asemenea fisier pe disc, tastati:

```
SUBMIT C SAMP1
```

-- si apasati pe CR.

Pentru informatii suplimentare despre fisierele SUBMIT consultati documentatia CP/M sau unul dintre manualele CP/M disponibile.

Compilarea sub M80 si L80 (Asamblori relocabili)

In linii generale procedura e aceeasi ca cea descrisa anterior, cu exceptie ca sint create module REL si apoi legate impreuna, dupa cum urmeaza:

```
CC SAMP1.C
C2 SAMP1.COD
M80 SAMP1,=SAMP1.ASM
L80 SAMP1,FUNC,CRUNT2,STDIO,ALLOC,C2RT,SAMP1/N/E:CCSTAR
```

Fiecare dintre comenzile de mai sus provoaca crearea unui fisier nou pe discul curent. Fisierele sint:

```
SAMP1.COD    : iesirea din CC
SAMP1.ASM    : iesirea din C2
SAMP1.REL    : iesirea din M80
SAMP1.COM    : iesirea din L80
```

Utilizatorii care folosesc procedura de mai sus trebuie sa aiba urmatarele fisiere in format REL:

```
ALLOC.REL    : de la ALLOC.C
STDIO.REL    : de la STDIO.C
CRUNT2.REL   : de la CRUNT2.C
FUNC.REL     : de la FUNC.C
C2RT.REL     : de la biblioteca de executie C2RT.MAC
```

Trebuie sa aveti pe disc si fisierul C2.RTM.

Un fisier SUBMIT corect pentru asamblorii relocabili este urmatorul:

```
CC $1.C
C2 $1.COD
M80 $1,=$1.ASM
L80 $1,CRUNT2,FUNC,STDIO,ALLOC,C2RT,$1/N/E:CCSTAR
```

Optiunile liniei de comanda CC

Optiunile sint urmatoarele (implicit e fara actiune, daca nu se specifica altceva):

-G Opreste generarea codului U. Aceasta optiune e utila pentru verificarea sintaxei unui fisier cod sursa. Alegerea acestei optiuni precum si optiunea +L de mai jos va crea o listare numerotata a codului sursa C cu expandarea macrodefinitiilor. Implicit se genereaza cod-U.

+CO Forteaza iesirea la consola. Este creat un fisier de iesire de lungime nula. Implicit e (-CO) pentru a genera "filename.COD".

+LNO Plaseaza numarul curent al liniei din fisierul sursa in fata primei linii corespunzatoare in cod-U din fisierul de iesire. Iesirea indica si numarul de linie la care incepe fiecare fisier incorporat prin utilizarea unei directive #INCLUDE. Sint indicate clar si directivele #INCLUDE consecutive.

+L Listeaza fiecare linie a codului sursa C sub forma unui comentariu in fisierul de iesire cod-U. Liniile sursa sint grupate impreuna dupa functii si sint astfel slab asociate cu liniile lor corespunzatoare in cod-U. Macro-urile sint expandate si numerele liniilor apar inaintea fiecarei linie cod sursa.

+F Scrie in fisier toate liniile in cod-U generate imediat dupa ce e prelucrata fiecare linie a codului sursa C. Daca sint alese ambele optiuni +L si +F, atunci fiecare linie de comentariu in cod sursa plasata in fisierul de iesire e urmata imediat de toate liniile cod-U generate din el. Alegerea acestei optini intirzie executarea lui CC si C2 si inhiba unele optimizari realizate in mod normal de C2.

+CR Lipeste un CR la fiecare linie de iesire generata in cod-U. Acest format e folosit de unele editoare de text (de ex. ED a lui CP/M)

+A Plaseaza adresa ultimei locatii de memorie folosita de CC in fata fiecarei linii cod sursa C, inserata ca un comentariu in fisierul de iesire. Aceasta optiune nu are efect daca nu a fost selectata si optiunea +L.

+DDT Incorporeaza informatiile suplimentare de depanare in fisierul de iesire.

-N Este imediat urmat de un intreg care specifica valoarea initiala pentru etichetele numerotate consecutiv, generate de CC. Implicit e o valoare initiala doi.

Optiunile liniei de comanda CC dependente de masina

Cele 3 optiuni prezentate in continuare pot fi folosite pentru a potrivi CC-ul pentru un procesor destinatie particular (CPU). Acestea sint singurele optiuni dependente de masina. Setarea implicita a optiunilor este corecta pentru seria 8080 a CPU-ului. Daca aceasta setare nu e potrivita pentru procesorul Dv. compilatorul nu va functiona corect (jflag, int-size si pad-size se vor referi la numele de variabile din compilator).

+J Seteaza Jflag egal cu 1. Ori de cite ori un singur octet reprezentind un char e trecut ca un parametru local la o functie , este extins la un int. Setarea lui jflag = 1 face ca un astfel de octet sa fie memorat in octetul cel mai din dreapta al locatiei asignate, fiind setarea corecta pentru Z8000. Implicit e setarea lui jflag cu 0 cauzind ca acelasi octet reprezentind un char sa fie memorat in octetul cel mai din stinga al locatiei asignate, fiind setarea

corecta pentru CPU-urile din seria 8080.

-I Este imediat urmat de un intreg care specifica marimea unui int (int-size) in octeti. Implicit e 2 octeti.

-P Este imediat urmat de un intreg care specifica modulul adecvat (pad-size) pentru toate adresele de intregi. Adica fiecare adresa de intreagi modulo pad-size va fi egala cu 0. Cu alte cuvinte, fiecare valoare intreaga va fi un multiplu de valoare pad-size.Implicit e un modul de 1.

Valorile pentru cele 3 optiuni de mai sus, necesare pentru a potrivi acest compilator pentru un procesor destinatie specifica pot fi setate intr-o operatie utilizind una dintre indicatiile (composite flags) definite in continuare.

+i8080 seteaza: jflag = 0, int-size = 2, pad-size = 1. Procesoarele destinatie sint: Intel 8080, 8085, 8086, 8088, 186, 188 si 286.

+Z80 seteaza aceleasi valori ca mai sus. Procesor destinatie e Zilog Z80.

+Z8000 seteaza : jflag = 1, int-size = 4, pad-size = 2. Procesorul destinatie e Zilog Z8001.

+Z8001 seteaza : jflag = 1, int-size = 4, pad-size = 2. Procesor destinatie Zilog Z8001.

+Z8002 seteaza : jflag = 1, int-size = 2, pad-size = 2. Procesor destinatie e Zilog Z8002.
Conditia implicita e identica cu rezultatul de la +i8080 sau +Z80.

Optiunile liniei de comanda C2

Optiunile sint urmatoarele (implicit e fara actiune, daca nu e specificat altcumva).

-G Opreste generarea codului de asamblare. Implicit e cu generarea codului de asamblare de iesire.

-O Opreste optimizarea. Este efectuata doar translarea cod-U in cod de asamblare. Implicit efectueaza optimizarea.

-Q Este imediat urmat de un intreg specificind nivelul de optimizare. Implicit e optimizarea totala. (Nu e implementat curent.)

-X Cauzeaza generarea unui cod putin mai mic in privinta vitezei de executie. Optimizarea realizata negociaza eficienta de viteza cu pretul eficientei de spatiu. Implicit e spatiu in favoarea vitezei. Diferenta e relativ mica pentru ca cele mai multe optimizari au un efect pozitiv asupra eficientei de viteza si spatiu.

+CO Forteaza iesirea la consola, rezultatul putind fi astfel examinat. Este creat doar un fisier de iesire de lungime nula. Implicit e iesirea la filename.ASM.

+L Plaseaza fiecare linie a codului U in fisierul de iesire (cod de asamblare) cu un comentariu. Liniile cod-U sint grupate dupa functii.

+PRGL Face ca numele functiei optimizate in mod curent sa fie afisat la consola ca o indicatie a locului unde C2 e in executie.

+OC Schiteaza procesul de optimizare scriind in fisierul de iesire fiecare schimbare de cod, in secventa in care e facuta. Aceasta optiune va incetini executia lui C2 si va spori mult cantitatea de informatie la iesire.

+T Genereaza si insereaza in fisierul de iesire secvente de cod care fac ca in timpul rularii sa apara la consola numele functiei apelate de program in acel moment. Aceasta optiune utila la depanare permite programatorului sa urmareasca fluxul de control din cadrul unui program. (Nu e curent implementat).

+Z E imediat urmat de un sir de caractere pe care le foloseste C2 ca un prefix pentru toate etichetele pe care le genereaza. Implicit e un prefix 'C'.

-RH<filename> Determina ca optimizorul sa foloseasca fisierul specificat ca fisier "header" de executie. (Obs. nu se pune spatiu intre -RH si <filename>).

-RT<filename> Determina ca optimizorul sa foloseasca fisierul specificat ca fisier "trailer" de executie. (Obs. nu se pune spatiu intre -RT si <filename>).

-ENT<keyword> Defineste cuvintul cheie folosit pentru a declara eticheta ca punct de intrare pentru asamblorii relocabili. (Obs. nu se pune spatiu intre -ENT si <keyword>).

-EXT<keyword> Defineste cuvintul cheie folosit pentru a declara eticheta externa pentru asamblorii relocabili. (Obs. nu se pune spatiu intre -EXT si <keyword>).

-ORGnnnn Determina codul de iesire sa inceapa la adresa specificata (ORG) in baza 10, prin nnnn.

Directivele preprocesorului

Toate intrarile fisierelor spre un compilator C SuperSoft trec printr-un preprocesor care cauta liniile incepind cu #. Asemenea linii, care nu sint ele insele compilate ci directioneaza un proces de compilare, sint cunoscute ca linii de control de compilare sau directive de preprocesare. Aceste linii se termina cu caractere de linie noua si nu cu punct si virgula. Directivele de preprocesor sint: #DEFINE, #INCLUDE si perechea #ASM si #ENDASM. Cu exceptia directivei #DEFINE, fiecare dintre aceste directive este sintactic independenta de restul limbajului C si poate sa apara oriunde intr-un program. Efectele tuturor directivelor se extind pina la sfirsitul unei compilari. Utilizarea si efectele acestor directive sint prezentate in continuare.

Directiva #DEFINE

Daca o linie de forma

 # DEFINE identifier token-string

se gaseste in sursa de intrare, aceasta obliga preprocesorul sa inlocuiasca toate aparitiile succesive ale identificatorului specificat (**identifier**) cu **token-string** (token = o unitate lexicala intr-un limbaj de programare, ca de ex. un identificator, un operator, un cuvint cheie, etc). Aceasta e cea mai simpla forma de substituire macro. Nu va fi inlocuit un **identifier** care apare intre apostroafe ('...') sau ghilimele ("...") intr-un #DEFINE. Preprocesorul va cauta in continuare alti identificatori in linia #DEFINE, creindu-se astfel posibilitatea folosirii directivelor #DEFINE incluse. Redefinirea unui identificator va duce la o eroare de compilare. Nu este inca implementata directiva #DEFINE parametrizata.

Cea mai obisnuita utilizare a acestei directive este definirea unui set de constante simbolice la inceputul programului, pentru utilizare in functiile care urmeaza (o practica mult mai buna decit inserarea constantelor literale in enunturile programului). Un exemplu prescurtat e urmatorul:

```
#DEFINE BSIZ 0x100
char Buf1[BSIZ], Buf2[BSIZ];
main()
{
        register int i;

        for(i=0;i++<BSIZ;){
                Buf1[i] = Buf[2] = 0;
```

Substituirea macro prin #DEFINE are multe alte utilizari; informatii si exemple sulplimentare gasiti in lucrarea lui Kernighan si Ritchie pag.12, 86 si 207.

Directiva #INCLUDE

Daca o linie de forma

#include "drivename:filename"
#include <drivename:filename>

e introdusa in compilator, ea obliga preprocesorul sa inlocuiasca aceasta linie cu intregul continut al fisierului (**filename**) de pe unitatea specificata (**drivename**). Daca preprocesorul nu poate gasi fisierul pe unitatea specificata, il va cauta pe discul curent. In viitor, (nu in versiunea actuala) preprocesorul va cauta fisierul numit pe toate unitatile de disc pe care le stie disponibile (toate unitatile mentionate in program pina la acel moment). Orice directiva #INCLUDE gasita intr-un fisier "#INCLUDE" va fi prelucrata in acelasi mod fiind deci posibila includerea directivelor #INCLUDE. Exemple si informatii suplimentare pot fi gasite in manualul lui Kernighan si Ritchie pag.86, 143 si 207.

Directivele #ASM, #ENDASM

Aceasta caracteristica speciala a limbajului C SuperSoft va ofera posibilitatea de a insera linii de cod de asamblare direct in fisierul cod sursa C. Se va prezenta in felul urmator:

```
.
.
.
putchar('y');
#asm

        mvi a,88
        call output

#endasm
orlf();
.
.
.
```

Toate liniile programului dintre #ASM si #ENDASM trec neschimbate prin

ambii moduli ai compilatorului si sint incorporate in locatile corespunzatoare din fisierul final de iesire. Nu sint nici optimizate si nici erorile nu sint cautate. In codul inserat nu trebuie sa folositi etichete care incep cu 'C', deoarece compilatorul genereaza si el etichete cu aceeasi litera de inceput, devenind astfel posibila multipla definire, ceea ce va duce la o eroare de asamblare.

Relocarea codului generat de compilator

Adresele de memorie dintr-un program asamblat cu un asamblor relocabil pot apare fie ca deplasari fata de o origine relativa a programului, fie ca referinte externe (adica o referinta facuta cu ajutorul unei etichete simbolice spre o locatie din alt program sau dintr-o zona externa de date). "Originea relativa a programului" este inceputul unui program oarecare, in legatura cu care sint calculate toate adresele sale interne. Aceste adrese ramin relative pina cind programul e incarcat la o anumita adresa absoluta care devine origine absoluta si toate celelalte adrese devin absolute cind sint calculate in raport cu ea.

Asamblorii relocabili permit unele mijloace de stabilire a locatiei absolute sau relative adecvate pentru programul Dv. In cazul asamblorului absolut ASM din CP/M poate fi folosita instructiunea ORG. Locatia unui ORG e folosita de ASM ca origine pentru codul pe care il asambleaza -- pina la urmatorul ORG sau sfirsit de fisier. Aceasta facilitate e deosebit de importanta atunci cind doriti sa incarcati programul in ROM, deoarece toate zonele de date in care va scrie programul din ROM trebuie sa fie in RAM. Argumentul instructiunii ORG specifica originea absoluta pentru codul generat de incarcatorul de programe al lui CP/M (LOAD.COM). Instructiunea ORG poate fi folosita si pentru a stabili diferitele origini relative si absolute pentru diferite segmente de program si zone de date. Astfel, daca programul pe care doriti sa-l incarcati in ROM solicita zone de date care pot fi scrise, cel putin doua enunturi ORG trebuie sa apara in fisier: unul la inceputul programului indicind originea absoluta in ROM si altul la inceputul zonei de date care pot fi scrise, specificind originea absoluta in RAM.

Iesirea finala in cod de asamblare generata de acest compilator cuprinde 2 enunturi ORG (la inceputul programului si al zonei de date), dupa cum urmeaza:

```
ORG *  ; REORIGIN PROGRAM HERE (localizare program)
.
.
.
ORG *  ; REORIGIN DATA HERE (localizare date)
.
.
.
```

Argumentul '*' reprezinta locatia curenta, ceea ce face ca un ORG astfel enuntat sa fie un "no-op" (nu se executa nici o operatie cind este intilnit). Aceste enunturi au efect doar cind inserati valorile argumentelor adecvate situatiei Dv.

Declararea functiilor de listare

In prezent, compilatorul de C SuperSoft are implementate functii de listare. Acestea sint functii fara un numar predeterminat de argumente. Functiile bibliotecii standard C SuperSoft au incluse urmatoarele functii de listare: **printf**, **scanf**, **fprintf**, **fscanf**, **sprintf** si **sscanf**.

Pentru o executie corecta, oricare functie de listare creata de Dv. trebuie sa fie declarata inainte de a fi apelata de program, in felul urmator:

extern type listf(.);

-- únde **listf** este o functie arbitrara de listare si **type** este o clasa arbitrara de memorare. Astfel, la fiecare invocare a acestei functii compilatorul C genereaza si impinge contorul de argumente ca ultim argument pe stiva de parametrii. Daca aceasta declaratie apare la inceputul programului ca modul extern global, el trebuie sa apara o singura data in program.

CAPITOLUL III

Functiile bibliotecii standard de C SuperSoft

Functiile bibliotecii standard implementate in compilatorul de C SuperSoft sint definite in urmatoarele fisiere:

```
C2.RH      : "header"-ul bibliotecii de executie
C2.RT      : "trailer"-ul biblitecii de executie
STDIO.C    : functiile standard de I/E gen UNIX
ALLOC.C    : functiile de alocare dinamica a memoriei
CRUNT2.C   : functiile auxiliare de executie
FUNC.C     : functii auxiliare
FORMATIO.C: printf si scanf
```

In timpul celui de al doilea pas, compilatorul incorporeaza automat in programul Dv. codul de asamblare precompilat si preoptimizat al bibliotecii de executie. Celelalte 3 fisiere contin definitiile celorlalte functii de biblioteca standard in cod sursa C SuperSoft.

Incorporarea functiilor bibliotecii stardand in program

Daca programul apeleaza o functie din ultimele 3 fisiere, trebuie sa introduceti definitia functiei in program inainte de compilare. Daca aceasta functie, la rindul ei, apeleaza alte functii, trebuie incorporate si definitiile lor. Exista 5 metode de incorporare a functiilor de C SuperSoft sau a segmentelor de cod in program. Puteti folosi primele 4 metode descrise in continuare pentru a incorpora in program orice cod C SuperSoft dorit, nu numai functiile bibliotecii standard.

Metoda 1-- Directiva #INCLUDE

Prima si cea mai simpla metoda este aceea de a folosi directiva de preprocesor #INCLUDE pentru a incorpora in program intregul fisier care contine functia sau functiile dorite. Optim e de a plasa orice directiva #INCLUDE pentru fisierele functiilor bibliotecii standard in "header"-ul programului, imediat dupa orice definitie de date externe.

```
FORMATIO.C foloseste STDIO.C
STDIO.C foloseste STDIO.H si ALLOC.C
ALLOC.C foloseste CRUNT2.C
FUNC.C sta singura.
```

Dezavantajul primar al includerii tuturor modulelor necesare e acela ca va poate crea un program prea vast, cuprinzind si functii care nu au fost niciodata apelate, cu un timp marit de compilare.

Metoda 2-- Listarea numelor fisierelor in linia de comanda a compilatorului CC

Daca listati un numar de nume de fisiere intr-o linie de comanda pentru primul pas al compilatorului, CC, el va analiza toate fisierele in ordinea listata ca si cum ar fi un singur fisier si va plasa rezultatul intr-un singur fisier de iesire. Astfel puteti incorpora in program orice fisier cod sursa C SuperSoft prin simpla listare a numelui sau in linia de comanda pentru CC (deoarece functiile pot sa apara in orice ordine intr-un fisier sursa C). Linia de comanda CC care va incorpora toate functiile bibliotecii standard in programul Y.C, este:

```
CC Y.C CRUNT2.C STDIO.C ALLOC.C FUNC.C FORMATIO.C
```

Aceasta metoda are toate avantajele si dezavantajele metodei anterioare, dar e mult mai flexibila. Pentru a schimba fisierele incorporate in program, si nu pentru a schimba programul, trebuie sa tastati doar un set diferit de nume de fisiere in linia de comanda.

Metoda 3 -- Precompilarea si inserarea in C2.RH

Unul sau toate fisierele de functii de biblioteca standard care sint in cod sursa C SuperSoft pot fi compilate, iar fisierul sau fisierele rezultate pot fi inserate, cu editare minima in biblioteca de executie a compilatorului, C2.RH. Procedeul e descris in detaliu in capitolul 4. Aceasta metoda mareste viteza cu care puteti compila programele care incorporeaza unul sau mai multe fisiere ale bibliotecii standard. Totusi, aceasta metoda prezinta acelasi dezavantaj al unei posibile marimi excesive de program ca si celelalte 2 metode descrise mai sus.

Un avantaj e acela ca indata ce v-ati format o idee despre setul sau seturile de functii folosite mai des in program, ea va poate ajuta sa va creati versiunea sau versiunile proprii de biblioteca de executie prin precompilarea si inserarea seturilor dorite in C2.RH (o metoda de generare a unui astfel de set de instructiuni este descrisa in general la metoda 4). Functiile astfel inserate pot fi alese fie din functiile bibliotecii standard, fie create pentru a va indeplini cerintele. Cu cit e mai mare experienta Dv. de programare, cu atit e mai buna alegerea bibliotecii sau bibliotecilor adaptate executiei. Cu aceasta metoda se poate economisi timp pentru compilarea programelor si se pot reduce la minim functiile inutile.

Metoda 4 -- "Cut and paste" (taie si lipeste)

Aceasta metoda e simpla in conceptie, dar laborioasa in executie. Ideea este de a crea, din fisierele de functii ale bibliotecii standard, un fisier sau fisiere care sa contina doar acele definitii de date si functii cerute de program ("cut") si de a imbina intr-un fel fisierul sau fisierele create in program inainte de a-l compila ("paste").

Avantajul acestei abordari e acela ca ambele coduri - sursa si obiect - vor fi atit de mici cit le va permite implementarea programului.

Dezavantajul e acela ca e sarcina Dv. de a puncta fiecare "i" si de a taia fiecare "t". Trebuie sa verificati ca sint prezente si intacte toate definitiile de date si functii necesare pentru executarea programului. Acest lucru e de durata si plictisitor.

Pe de alta parte, pentru unele din aplicatiile Dv. va fi esentiala utilizarea cit mai eficienta a memoriei disponibile. Marimea unor programe poate sa atinga sau sa depaseasca capacitatea de memorie a sistemului Dv., in timp ce altele pot necesita spatii pentru memorarea datelor in timpul executiei.

Uneori puteti constata ca folositi repetat un subset particular de functii de biblioteca standard amestecate cu unele functii proprii. In acest caz, puteti folosi aceasta metoda ca o prima etapa in crearea setului sau seturilor "proprii".

Nu trebuie sa incercati sa folositi aceasta metoda daca nu sinteti familiarizati cu limbajul C si sistemul Dv. de operare. Ceea ce urmeaza e doar o trecere in revista a etapelor metodei "cut and paste".

i. Faceti o copie pe disc a fiecarui fisier care contine definitiile datelor si functiilor pe care doriti sa le incorporati in program.
ii. Folosind editorul, stergeti din fiecare fisier acele definitii de date si functii pe care nu doriti sa le introduceti in program. Fiti atenti sa nu stergeti definitiile acelor functii apelate de functiile apelate de program sau oricare dintre definitiile de date cerute de oricare functie pe care doriti sa o incorporati.
iii. Daca aveti mai mult de un fisier care contine definitiile necesare, le puteti lasa ca fisiere separate sau le puteti concatena.

iv. Sinteti pus din nou in situatia de a alege modul in care sa va introduceti fisierele in program. Puteti folosi oricare dintre cele 3 metode descrise anterior deoarece fiecare metoda merge pentru oricare fisier. Un avantaj suplimentar e acela ca programele Dv. vor contine acum doar definitiile de date si functii pe care le solicita. Restul va fi indepartat. O a 4-a optiune e aceea de a folosi editorul pentru a insera definitiile fisierului sau fisierelor pe care le ati creat in program, la destinatiile potrivite.

Metoda 5 -- Asamblori relocabili

Utilizatorii cu asamblori relocabili, ca de ex. pachetul M80-L80 Microsoft, pot lega simplu in program rutinele necesare. Procedura e de a compila separat functiile sau grupurile de functii , si apoi de a le uni intr-un tot.(Vezi capitolul II , Instructiuni de operare).

Limbajul de C SuperSoft permite directiva EXTERN pentru variabilele externe, adica variabile globale. Deasemenea, toate functiile nedefinite sint automat declarate ca variabile externe cu toate referintele rezolvate in momentul unirii.

Relocatarea e pe departe cea mai avantajoasa metoda de incorporare a functiilor utilizator si de biblioteca intr-un fisier final COM. Totusi, aceasta optiune e oferita doar utilizatorilor prevazuti cu pachet de asamblori relocabili.

Alegerea metodei de lucru

Experienta Dv. ca programator in limbaj C va va indica ce metoda sa alegeti. Programatorii incepatori vor utiliza fara indoiala Metoda 1, deoarece ofera cele mai putine posibilitati de a avea dificultati. Chiar si programatorii experimentati vor alege Metoda 4 doar in zilele lor bune. Metoda 5 este cea mai buna, dar e doar pentru sistemele prevazute cu asamblori relocabili. In concluzie, alegeti metoda care vi se va parea ca foloseste mai eficient timpul Dv. de programare pentru o situatie specifica.

Initializarea functiilor de alocare dinamica a memoriei

Este **esential** ca programul Dv. sa initializeze functia de alocare a memoriei dinamice - **alloc** - inainte de a o apela direct sau indirect. Daca nu se realizeaza acest lucru, functiile **alloc** si **free**, precum si toate functiile care le apeleaza nu vor functiona corect. Apelarile indirecte la **alloc** se pot face prin apelari la functiile pe care le-ati definit in program sau prin apelari la functiile I/E ale fisierelor bufferate definite in STDIO.C.

Daca programul apeleaza **alloc** direct sau indirect, printr-un apel la o functie definita in cadrul programului, trebuie sa declarati o variabila externa (implicita), **"allocs"**, de tip pointer la un caracter (char *) si sa-i atribuiti un 0 inainte de a face apelarea directa sau indirecta la **alloc**. Vezi descrierea functiei **alloc** de la inceputul cap. III in privinta succesiunii recomandate de enunturi.

Daca programul include (prin directiva #INCLUDE) fisierul STDIO.C cu functii de biblioteca standard si apeleaza functiile tip buffer I/E pe care le defineste (care la rindul lor apeleaza **alloc**), trebuie sa stabiliti setarea lui **allocs** la zero (0) inaintea oricarei apelari directe sau indirecte la **alloc**, inclusiv acelea facute la functiile de I/E ale fisierului tip buffer.

Descrierea functiilor bibliotecii standard de C SuperSoft

In prezentarea acestor functii s-a adoptat o prescurtare pentru indicarea tipului de date intoarse de functie, sau date functiei ca argumente. Cele citeva rinduri care apar inaintea descrierii fiecarei functii sint primele rinduri ale definitiei in limbaj C SuperSoft, si anume, declaratiile functiei in sine si ale

argumentelor ei. Tipul unei functii e tipul valorii pe care o intoarce. Daca nu e indicat tipul unei functii, atunci aceasta nu va intoarce nici o valoare definita. Tipul fiecaruia dintre argumentele sale va fi intotdeauna indicat.

OBS. Fisierul STDIO.H contine instructiuni #DEFINE care stabilesc urmatoarea pereche de valori simbolice de intoarcere predefinite (valorile numerice reale intoarse sint date in paranteze): SUCCEES (0) si ERROR (-1). Multe dintre functiile definite in STDIO.C vor intoarce aceste valori simbolice, folosite pentru a intelege mai usor codificarea acestor functii.

STDIO.C include (prin directiva #INCLUDE) fisierul STDIO.H.

ABS

int abs(i)

int i;

abs intoarce valoarea absoluta a lui **i**. (**abs** intoarce **-i**, daca **i**<0, altfel intoarce **i**).

ALLOC

char *alloc(n)

int n;

alloc aloca o zona contigua de memorie de lungime **n**. Fiecare bloc pe care il aloca incepe la <u>adresa para</u>.

alloc, daca reuseste, intoarce un pointer la prima locatie de memorie din bloc. Trebuie sa memorati acest pointer pentru utilizari ulterioare in program. **alloc** intoarece o valoare de pointer nula (0) si nu aloca nici o zona de memorie daca alocarea unui bloc contiguu de marimea ceruta s-ar suprapune peste:

1) un program memorat,
2) stiva de executie, sau
3) un bloc alocat anterior in memorie. (Vezi K&R, pag. 97).

OBS. Pentru ca **alloc** sau pentru ca orice alta functie care apeleaza **alloc** sa functioneze corect programul trebuie sa stearga (asignati-i o valoare 0) variabila externa **allocs** de tip pointer la un caracter (char *) inainte de primul apel a lui **alloc** sau a oricarei functii care apeleaza **alloc**. Acest lucru face ca **alloc** sa se initializeze la primul apel. **allocs** e declarat in ALLOC.C.

ATOI

int atoi(s)

char *s;

atoi intoarce intregul in baza zece corespunzator sirului ASCII terminat in zero indicat de **s**. Daca acest sir contine alte caractere decit tab-urile frontale, blank-urile si un semn minus (toate optional) urmate de cifre zecimale consecutive, **atoi** intoarce 0. (Vezi K&R, pag.39,58)

BDOS

```
int bdos(c,de)

        int c, de;
```

bdos inlesneste utilizatorilor CP/M sa incorporeze apelarile de functii directe BDOS in programele scrise in limbaj C. Programele care folosesc **bdos** nu vor fi portabile decit sub CP/M si CPU din seria 8080. **bdos** seteaza registrul C al masinii la valoarea data in **c** si perechea DE la valoarea din **de**, apoi initiaza un apel BDOS. BDOS se asteapta ca registrul C sa contina un numar valid de functie BDOS. Pentru informatii suplimentare consultati manualul de CP/M anexat echipamentului Dv.

bdos intoarce un singur octet (ca intreg) identic cu continutul registrului A prin care BDOS intoarce valoarea. Aceasta valoare intoarsa de **bdos** nu are extensie de semn.

BIOS

```
int bios(jmpnum,bc,de)

        int jmpnum, bc, de;
```

bios inlesneste utilizatorului CP/M sa incorporeze apelurile directe BIOS in programele scrise in limbaj C. Programele care apeleaza **bios** nu vor fi portabile decit sub CP/M si CPU din seria 8080. **bios** seteaza perechea BC a registrului masina la valoarea data in **bc** si perechea DE a registrului la valoarea din **de**, initializind apelul corect BIOS prin transferarea controlului la punctul de intrare a vectorului de salt BIOS specificat in **jmpnum**. Acest punct poate fi specificat numeric sau simbolic.

Mnemonicele sau numele simbolice pentru fiecare punct de intrare sint urmatoarele:

```
WBOOT ----  0        SELDSK ---  8
CONST ----  1        SETTRK ---  9
CONIN ----  2        SETSEC --- 10
CONOUT ---  3        SETDMA --- 11
LIST -----  4        READ ----- 12
PUNCH ----  5        WRITE ---- 13
READER ---  6        LISTST --- 14
HOME -----  7        SECTRAN -- 15
```

Daca **jmpnum** este fie SELDSK (8) fie SECTRAN (15), **bios** intoarce valoarea ramasa in registrul HL dupa executarea apelului BIOS; altfel, intoarce valoarea ramasa in registrul A fara extensie de semn (ca o valoare intre 0 si 255).

BRK

```
char *brk(p)

        char *p;
```

brk seteaza variabila externa CCEDATA in fiecare octet indicat de **p** si intoarce CCEDATA care e echivalenta cu valoarea pointerului trimis. CCEDATA e initial setat in octetul imediat urmator ultimului octet din zona externa de date a programului. Deoarece CCEDATA e folosit ca o valoare de baza de catre alte functii de alocare dinamica a memoriei, se realizeaza tot odata si initializarea corecta a acestor functii ce vor fi ulterior necesare programului Dv. Aproape niciodata nu aveti nevoie de apelarea lui **brk** in programul Dv.

CCALL

int ccall(addr,hl,a,bc,de)

char *addr, a;
int hl, bc, de;

ccall seteaza registrele de masina HL, A, BC si DE la valorile date in **hl**, **a**, **bc** si **de**, si apeleaza subrutina in limbaj de asamblare incepind de la **addr**. **ccall** intoarce valoarea prezenta in registrul HL dupa executia subrutinei. Programele care apeleaza **ccall** nu vor fi portabile decit sub CPU din seria 8080.

CCALLA

int ccalla(addr,hl,a,bc,de)

char *addr, a;
int hl, bc, de;

ccalla e identic cu **ccall** cu exceptia ca acesta intoarce valoarea prezenta in registrul A dupa executia subrutinei. Programele care apeleaza **ccalla** nu vor fi portabile decit sub CPU din seria 8080.

Ca exemplu al utilizarii acestei functii luati in considerare urmatoarele:

```
int bdos(c,de)
 char c;
 int de;
{
        return ccalla(5,0,0,c,de);
}
```

Functia C prezentata mai sus e evident o implementare a functiei BDOS folosind **ccalla**.(BDOS nu e implementat in acest mod). Atit **ccall** cit si **ccalla** pot fi folosite pentru a apela subrutine in limbaj de asamblare create de Dv.

CLOSE

int close(fd)

FILE *fd;

close inchide fisierul specificat de pointerul **fd** descriptorului sau de fisier. **close** intoarce SUCCESS (0) daca fisierul specificat a fost inchis, sau intoarce ERROR (-1) si nu inchide fisierul, daca:

1) **fd** nu indica un descriptor valid de fisier, sau

2) fisierul nu poate fi inchis datorita unei erori la nivel de sistem de operare.

close nu plaseaza un caracter "end-of-file" (EOF) in buffer (daca exista unul asociat fisierului) si nu apeleaza pe **fflush**. Daca apelati **close** pentru un fisier deschis prin **fopen** cu iesire buffer, fara ca mai intii sa-l apelati pe **fflush** , veti pierde orice date ramase in bufferul acelui fisier I/E. (Vezi K&R, pag. 163).

CODEND

char *codend()

codend intoarce un pointer in octetul imediat urmator sfirsitului de cod pentru segmentul radacina din programul Dv. Daca n-ati relocat zona de date externe din programul Dv., valoarea intoarsa de **codend** va indica inceputul acelei zone. (Nu e implementat curent).

CPMVER

int cpmver()

cpmver intoarce valoarea prezenta in registrul HL dupa executia unui apel al functiei READ (functie BDOS). Aceasta valoare de intoarcere, care nu are extensie de semn, este:
- 0, daca programul apelat ruleaza sub versiunile CP/M aparute inainte de versiunea 2.0,
- 0x0020, daca ruleaza sub CP/M versiunea 2.0,
- in gama 0x0021 - 0x002F sub versiunile care urmeaza dupa 2.0, si
- 0x0100 sub MP/M.

Aceasta functie e utila la scrierea programelor C pentru a rula sub CP/M sau MP/M indiferent de numarul de versiune. Programele care apeleaza **cpmver** nu vor fi portabile decit sub CP/M sau MP/M.

CREAT

FILE *creat(fspec)

FILESPEC *fspec;

creat creaza un fisier pe disc cu specificatia de fisier data in **fspec** si il deschide pentru iesirea directa. Va fi sters orice fisier cu aceeasi specificatie, existent pe disc.

creat, daca reuseste, intoarce un pointer la un descriptor valid de fisier pentru fisierul specificat. Trebuie sa memorati acest pointer pentru utilizari ulterioare in program. **creat** intoarce ERROR (-1) si nu creaza sau nu deschide nici un fisier, daca:

1) nu exista suficienta memorie de utilizator pentru a memora un descriptor nou de fisier,
2) specificatia fisierului dat e invalida, sau
3) fisierul nu poate fi creat sau deschis datorita unei erori la nivel de sistem de operare. (Vezi K&R, pag. 162)

ENDEXT

char *endext()

endext intoarce un pointer la octetul imediat urmator ultimului octet din zona de date externe a programului Dv. Aceasta valoare ar trebui sa fie identica cu valoarea initiala a lui CCEDATA. (Nu e implementat curent).

EVNBRK

```
char *envbrk(n)
     int n;
```

envbrk functioneaza identic cu **sbrk** cu exceptia ca intotdeauna intoarce o valoare para. **evnbrk** realizeaza acest lucru prin "salt" peste un octet daca **sbrk** ar intoarce o valoare para pentru argumentul **n** dat. Daca reuseste, **evnbrk** va intoarce un pointer la prima locatie de memorie in blocul adaugat. **evnbrk** intoarce o valoare -1 si nu adauga nici un octet la memoria utilizatorului daca adaugarea numarului de octeti specificati se va:

1) suprapune peste un program memorat,
2) suprapune peste stiva de executie, sau
3) depaseste memoria disponibila.

IMPORTANT: Apelarea lui **evnbrk** cu un argument negativ nu trebuie sa fie facuta intre apelarile lui **alloc**.

EXEC

```
int exec(fspec)

     FILESPEC *fspec;
```

exec incarca fisierul specificat in sirul terminat in zero indicat de **fspec** si executa codul executabil care e presupus ca e continut de fisier. (O constanta de sir, ca de ex. "nextprog", poate fi introdusa in **fspec** deoarece evalueaza un pointer la un sir terminat in zero.) Daca nu e specificata nici o extensie sau tip de fisier, **exec** adauga ".COM" la numele fisierului specificat inainte de cautarea fisierului.

exec intoarce valoarea -1 daca nu exista fisierul sau nu poate fi deschis sau citit datorita unei erori la nivelul sistemului de operare. Daca reuseste, fisierul va fi incarcat peste programul care l-a apelat si nu va fi posibila intoarcerea la acel program. Totusi, printr-o alta apelare a lui **exec** se poate re-suprapune programul initial peste cel incarcat anterior. **exec** ofera posibilitatea de a inlantui sau executa succesiv o serie de fisiere de program, fiecare din serie suprapunindu-se peste programul precedent care l-a apelat.

Datele pot fi trecute din programul apelant in programul apelat fie prin fisiere, fie prin zona de date externe. Pentru a trece mai eficient datele in cadrul unui fisier, programul apelant trebuie sa inchida fisierul iar programul apelat trebuie sa il redeschida. In cazul datelor trecute prin zona de date externe, originea acelei zone trebuie sa fie aceeasi pentru ambele programe.

EXECL

```
int execl(fspec,arg1,arg2,...,0)

     FILESPEC *fspec;
     char *arg1, *arg2, ...;
```

execl e identic cu **exec**, cu exceptia ca parametrii liniei de comanda pot fi trimisi la programul apelat intr-o serie de siruri terminate in zero indicate de **arg1**, **arg2**, ... Ultimul argument trimis trebuie sa fie zero. **execl** construieste o linie de comanda din sirurile indicate de argumentele sale, lasind spatii intre ele.

Ca o alternativa, sub CP/M intreaga linie de comanda poate fi trecuta in sirul indicat de **arg1**. Spatiile trebuie sa apara in locurile adecvate in acel sir, iar **arg2** trebuie sa fie 0.

In ambele cazuri, parametrii trebuie sa apara, in ordine, in sir sau siruri asa cum ar aparea daca ar fi tastati intr-o linie de comanda pentru programul

apelat.(Constantele de sir pot fi folosite in locul pointerilor de sir pentru **arg1**, **arg2** , ..., precum si pentru **fspec**). CP/M cere ca lungimea liniei de comanda construita sa nu depaseasca 126 caractere. Alte sisteme de operare pot sa impuna mai multe sau mai putine restrictii.

EXIT

exit()

exit transfera controlul de la program inapoi in sistemul de operare. **exit** nu scrie pe disc nici un buffer de iesire (adica nu face **flush**) si nici nu inchide vreun fisier deschis. (Vezi K&R, pag. 154).

EXTERNS

char *externs()

externs intoarce un pointer la primul octet din zona de date externe a programului Dv. Daca nu ati relocat aceasta zona, aceasta valoare va fi identica cu cea returnata de **codend.** (Nu e implementat curent.)

FABORT

int fabort(fd)

FILE *fd;

fabort elibereaza descriptorul de fisier alocat unui fisier deschis, fara sa inchida acel fisier. Fisierul e specificat de pointerul **fd** spre descriptorul sau de fisier. Apelarea lui **farbort** nu influenteaza continutul unui fisier deschis doar pentru intrare, in schimb, apelarea lui pentru un fisier deschis pentru iesire poate provoca pierderea unora sau tuturor datelor scrise in acel fisier. (Am inclus aceasta functie datorita compatibilitatii cu BDS C, dar nu recomandam utilizarea ei).

fabort intoarce SUCCESS (0), daca reuseste, si ERROR (-1), daca **fd** nu indica un descriptor valid de fisier.

FCLOSE

int fclose(fd)

FILE *fd;

fclose inchide un fisier deschis cu **fopen** pentru o iesire la nivel de buffer. Fisierul e specificat de pointerul **fd** spre descriptorul sau. **fclose** plaseaza un caracter EOF in pozitia curenta din bufferul I/E al fisierului si apeleaza **fflush** inainte de a inchide fisierul.

fclose intoarce SUCCESS (0), daca fisierul specificat a fost inchis cu succes, sau ERROR (-1) si nu inchide fisierul daca:

1) fisierul specificat nu a fost deschis pentru iesirea la nivel de buffer prin **fopen**,

2) **fd** nu indica un descriptor valid de fisier, sau

3) fisierul nu poate fi inchis datorita unor erori la nivel de sistem de operare.

FFLUSH

```
fflush(fd)

        FILE *fd;
```

fflush scrie in fisier continutul curent al bufferului I/E asociat fisierului deschis pentru iesirea la nivel de buffer prin **fopen.** Fisierul e specificat de pointerul fd spre descriptorul sau de fisier. Marimea bufferului I/E, care e setat cind e deschis fisierul, e un multiplu intreg pozitiv al marimii inregistrarii de sistem. (O inregistrare de sistem e unitatea minima a datelor transferate in timpul operatiilor de I/E ale fisierului. Marimea de inregistrare sub CP/M e de 128 octeti. Sub UNIX e 1 octet. Consultati documentatia sistemului de operare pentru informatii suplimentare.) Dupa o apelare la **fflush** pointerul I/E al fisierului va indica inceputul inregistrarii care urmeaza ultimei inregistrari scrise.

fflush intoarce SUCCESS (0), daca bufferul a fost scris bine in fisier (apelarea lui **fflush** cind bufferul e gol nu are alt efect decit sa intoarca SUCCESS), si intoarce ERROR (-1) si nu scrie bufferul daca:

1) fisierul nu a fost deschis pentru iesirea la nivel de buffer prin **fopen**,
2) fd nu indica un descriptor valid de fisier, sau
3) n-a putut fi scris tot bufferul din cauza unei erori la nivelul sistemului de operare. (Vezi K&R, pag. 166)

OBS. Deoarece bufferul de iesire al fisierului e scris automat ori de cite ori e plin, de fiecare data cind apelati **fflush**, bufferul va contine unii octeti de date pe care doriti sa-i salvati si unele resturi (octeti nedefiniti). Daca sinteti la sfirsitul unui fisier puteti sa scrieti un caracter EOF ca ultim octet de date inainte de apelarea lui **fflush** si de inchiderea fisierului. (prin **fclose**). De asemenea, puteti cauta si citi orice inregistrare care a fost scrisa, dupa apelarea lui **fflush**, dar inainte de inchiderea fisierului. Totusi, apar probleme daca nu sinteti la sfirsitul fisierului si doriti sa cautati o alta inregistrare: deoarece pentru orice fisier deschis pentru iesirea la nivel de buffer trebuie sa apelati **fflush** inaintea oricarei apelari a lui **seek** - pentru a nu pierde continutul bufferului - nu puteti evita descarcarea resturilor in mijlocul fisierului.

FGETS

```
char *fgets(s,n,fd)

        char *s;
        int n;
        FILE *fd;
```

fgets citeste maximum n-1 caractere (octeti) din fisierul deschis cu **fopen** (pentru intrare la nivel de buffer) in sirul care incepe la **s.** Fisierul e specificat de pointerul **fd** spre descriptorul de fisier. **fgets** va opri citirea fie la intilnirea unui caracter de linie noua (\n), fie dupa n-1 octeti. **fgets** adauga apoi un caracter nul la caracterele citite pentru a crea un sir terminat in zero.

fgets, daca reuseste, intoarce un pointer la acest sir (identic cu valoarea care i-a fost trimisa in **s**) si o valoare de pointer zero daca:

1) **fd** nu indica un descriptor valid de fisier,
2) fisierul nu poate fi citit datorita unei erori la nivel de sistem de operare, sau
3) s-a ajuns la sfirsitul fisierului. (Vezi K&R, pag. 155)

FOPEN

```
FILE *fopen(fspec,mode,buffer-size)

     FILESPEC *fspec;
     char *mode;
     int buffer-size;
```

fopen creaza si/sau deschide un fisier pentru I/E la nivel de buffer cu specificatia de fisier data in **fspec**. Daca **fopen** reuseste, intoarce un pointer spre un descriptor valid de fisier pentru fisierul specificat. Trebuie sa memorati acest pointer pentru utilizari ulterioare in program. **fopen** intoarce ERROR (-1) si nu creaza, nici nu deschide un fisier daca:

1) valoarea specificata pentru **buffer-size** e mai mica decit marimea inregistrarii de sistem (128 octeti in CP/M)
2) nu e disponibila o memorie utilizator sufucienta pentru o noua inregistrare,
3) specificatia fisierului e invalida, sau
4) fisierul nu poate fi deschis sau creat datorita unei erori la nivelul sistemului de operare.

Pentru **mode** trebuie sa specificati "w" ,"a", "r" sau "rw", determinind astfel modul de I/E al fisierului, conform tabelului urmator:

"w"	mod doar de scriere
"a"	mod doar de scriere, fisierul se completeaza prin adaugiri succesive, pina la intilnirea EOF
"r"	mod doar de citire
"rw"	mod de scriere-citire

Daca modul I/E al unui fisier e "w", "a" sau "rw", se spune ca e deschis pentru iesirea la nivel buffer. Daca modul I/E al unui fisier e "r" sau "rw" se spune ca e deschis pentru intrarea la nivel buffer.

Valoarea pe care o specificati pentru **buffer-size** determina marimea bufferului I/E asociat fisierului. Aceasta valoare trebuie sa fie un multiplu intreg pozitiv al marimii inregistrarii de sistem. (O inregistrare de sistem e unitatea minima de date transferate in timpul operatiilor I/E de fisier. Sub CP/M, marimea e de 128 octeti. Sub UNIX este un octet. Consultati documentatia sistemului de operare pentru informatii suplimentare). Un pointer catre primul octet din acest buffer I/E e memorat in descriptorul de fisier. Astfel, doar un pointer catre descriptorul de fisier trebuie sa fie trimis spre oricare dintre celelalte functii I/E ale fisierului.(Vezi K&R, Pag 151,167)

FPRINTF

```
int fprintf(fd,format,arg1,arg2,...)

     FILE *fd;
     char *format;
     ...
```

fprintf este identic cu **printf** cu exceptia ca in loc de a scrie la iesirea standard, isi scrie sirul de iesire formatat in bufferul I/E asociat unui fisier deschis pentru iesirea la nivel de buffer prin **fopen**, incepind de la locatia curenta din acel buffer. Bufferul este automat scris in fisier ori de cite ori se umple). Fisierul e specificat de pointerul **fd** spre descriptorul sau.

fprintf e o functie de listare care trebuie sa fie declarata inainte de utilizare. (Vezi cap. II, Declararea functiilor de listare).

fprintf intoarce SUCCESS (0) daca a reusit scrierea intregului sir de iesire. Totusi, datorita bufferarii operatiilor I/E ale fisierului, o astfel de valoare de intoarcere nu poate garanta ca acelasi sir identic va fi scris cu succes in fisier, deoarece erorile care rezulta din, si care afecteaza executia unei anumite apelari a lui **fprintf** pot sa nu iasa in evidenta pina cind o functie apelata ulterior face ca bufferul I/E al fisierului sa fie scris.

fprintf intoarce ERROR (-1) daca:

1) **fd** nu indica un descriptor valid de fisier,

2) fisierul nu a fost deschis pentru iesirea la nivel de buffer prin **fopen**,

3) fisierul nu a putut sa fie scris datorita unei erori la nivelul sistemului de operare, sau

4) nu s-a putut scrie tot sirul in fisier datorita lipsei de spatiu pe disc. (Vezi K&R, pag. 152)

FPUTS

```
int fputs(s,fd)

        char *s;
        FILE *fd;
```

fputs scrie sirul terminat in zero indicat de s in bufferul I/E asociat unui fisier deschis pentru iesirea la nivel de buffer prin **fopen**, incepind de la locatia curenta din acel buffer. Bufferul este automat scris cind se umple (adica, intregul sau continut e scris in fisier). Fisierul e specificat de pointerul **fd** spre descriptorul de fisier. Pentru fiecare caracter de linie noua ('\n')care apare in sir sint scrise in buffer un CR si o linie noua ("\r\n"). Nu este scris caracterul terminal nul.

fputs intoarce ERROR (-1) si nu scrie sirul daca:

1) **fd** nu indica un descriptor valid de fisier,

2) fisierul nu a fost deschis pentru iesirea la nivel de buffer prin **fopen**, sau

3) fisierul nu a putut sa fie scris datorita unei erori la nivel de sistem de operare.

Altfel, **fputs** intoarce numarul de octeti scrisi in buffer minus numarul de caractere CR inserate. Totusi, din cauza bufferarii operatiilor I/E ale fisierului, o astfel de valoare de intoarcere nu garanteaza ca vor fi scrisi in fisier aceeasi octeti, deoarece erorile care rezulta si care afecteaza iesirea unui anumit apel la **fputs** nu pot sa iasa in evidenta pina la o apelare ulterioara a unei functii care face ca bufferul sa fie scris in fisier. (Vezi K&R, pag. 155)

FREE

```
free(p)

        char *p;
```

free elibereaza un bloc alocat anterior in memorie printr-un apel la **alloc**. Argumentul **p**, care trebuie sa fie identic cu valoarea intoarsa de apelarea lui **alloc**, este un pointer spre prima locatie de memorie din bloc. Blocurile alocate pot fi eliberate in orice ordine. Apelarea lui **free** cu un argument care nu a fost obtinut anterior prin apelarea lui **alloc** este o eroare grava. (Vezi K&R, pag. 97, 177)

FSCANF

int fscaf(fd,format,arg1,arg2,...)

FILE *fd;
char *format, *arg1, *arg2, ...;

fscanf e identic cu **scanf** cu exceptia ca sirul de intrare e citit din bufferul I/E asociat cu un fisier deschis pentru intrarea la nivel de buffer prin **fopen** si nu din intrarea standard. Fisierul e specificat de pointerul **fd** spre descriptorul de fisier. **fscanf** incepe citirea la pozitia curenta din bufferul I/E si opreste citirea cind a asignat corect valorile octetilor corespunzatori fiecarui articol listat in sirul **format**, sau cind s-a ajuns la sfirsitul fisierului.

fscanf este o functie de listare care trebuie sa fie declarata inainte de utilizare. (Vezi cap.II)

Daca nu apar erori, **fscanf** intoarce numarul de valori asignate cu succes; intoarce ERROR (-1) si nu executa intrari daca:

1) **fd** nu indica un descriptor valid de fisier,
2) fisierul nu a fost deschis pentru intrarea la nivel de buffer prin **fopen**, sau
3) fisierul nu a putut fi citit datorita unei erori la nivel de sistem de operare. (Vezi K&R, pag. 152)

GETC

int getc(fd)

FILE *fd;

getc intoarece un caracter (octet) ca un intreg (intre 0 si 225 inclusiv) in secventa din fisierul deschis pentru intrarea la nivel de buffer prin **fopen**. Fisierul e specificat de pointerul **fd** spre descriptorul de fisier. CR si avansuri la linie noua (LF) sint intoarse explicit. **getc** intoarce ERROR (-1) daca:

1) fisierul nu a fost deschis pentru intrarea la nivel de buffer, sau
2) s-a ajuns la sfirsitul fisierului. (Vezi K&R, pag. 152,166)

GETCHAR

char getchar()

getchar intoarce urmatorul caracter din intrarea standard (CP/M dispozitivul CON: de obicei tastatura consolei). Daca **getchar** intilneste un CTRL-Z (EOF sub CP/M) intoarce un caracter nul. Atunci cind un program e rulat sub CP/M si intilneste un CTRL-C, va fi abandonat si controlul va fi intors la rutina care a initiat acest program. (Vezi K&R, pag. 13,40,144,152,161,162)

GETS

gets(s)

char *s;

gets citeste linia urmatoare din intrarea standard (CP/M dispozitivul CON: de obicei tastatura consolei) in sirul incepind la **s**. **gets** inlocuieste caracterul linie noua (\n) sau combinatia CR,LF (\r\n) care termina introducerea liniei cu caracterul nul pentru a crea un sir terminat cu zero. Deoarece **gets** nu testeaza daca sirul incepind la **s** este destul de lung pentru a contine toata linia de intrare, trebuie sa definiti acest sir astfel incit sa poata sa contina

cea mai lunga linie de intrare posibila. (Vezi K&R, pag. 155)

GETVAL

int getval(s)

char **s;

getval trebuie trimis ca un pointer la un pointer spre un sir de caractere ASCII terminat in zero, format din subsiruri separate prin virgule. Fiecare subsir trebuie sa contina doar blank-uri initiale,tab-uri sau semn minus (toate optional) urmate de cifre zecimale consecutive.

Cind **getval** e apelat initial cu un parametru care satisface cerintele mentionate mai sus, intoarce valoarea intregului in baza 10 corespunzatoare primului subsir si incrementeaza pointerul indicat de acel parametru (pointerul secundar) astfel incit acesta va indica primul caracter din subsirul urmator. Fiecare apelare ulterioara a lui **getval** cu acelasi parametru intoarce valoarea intregului in baza 10 corespunzatoare subsirului curent indicat de pointerul secundar si incrementeaza acel pointer astfel ca va indica primul caracter din subsirul urmator.

Deoarece **getval** intoarce un intreg binar de 16 biti cu semn, valoarea reprezentata in orice subsir nu trebuie sa fie mai mare de +32767 si nici mai mica de -32768.

Daca **getval** intilneste un caracter invalid la inceputul unui subsi, , el intoarce o valoare zero si incrementeaza pointerul secundar ca mai inainte. Daca **getval** intilneste un caracter invalid in cadrul unui subsir, el termina subsirul, intoarce valoarea corespunzatoare caracterelor valide citite pina la acel punct si incrementeaza pointerul secundar ca mai inainte. Cind **getval** intilneste caracterul terminal nul, el seteaza pointerul secundar la zero si intoarce o valoare zero.

GETW

int getw(fd)

FILE *fd;

getw intoarce un intreg, in secventa, dintr-un fisier deschis pentru iesirea la nivel de buffer prin **fopen**. Fisierul e specificat de pointerul **fd** catre descriptorul de fisier. CR si avansurile la linie noua (LF) sint intoarse explicit. **getw** intoarce ERROR (-1) daca:

1) fisierul nu a fost deschis pentru intrarea la nivel de buffer,
2) s-a ajuns la sfirsitul fisierului, sau
3) intregul -1 apare in fisierul de intrare.

De aceea eroarea trebuie verificata pentru a depista adevaratele conditii de eroare. Apelari la **getc** pot fi intercalate cu apelari la **getw**. Informatiile citite de **getc** si **getw** pot fi scrise de **putc** si **putw**. Nu e necesara nici o aliniere speciala a informatiilor in fisierul de intrare.

INDEX

char *index(s,c)

char *s, c;

index intoarce un pointer spre prima aparitie a caracterului **c** in sirul care incepe la **s**. **index** intoarce o valoare de pointer nula (0) daca **c** nu apare in sir.

INITB

```
initb(array,s)

     char *array, *s;
```

initb permite initializarea relativ convenabila a tablourilor de caractere. Trebuie trimisi 2 parametri: primul, **array**, trebuie sa fie un pointer la un tablou de caractere iar al doilea, **s**, trebuie sa fie un pointer la un sir de caractere ASCII terminat in zero, reprezentind valori intregi in baza 10, separate de virgule. Cind e apelat, **initb** transforma succesiv fiecare valoare intreaga in baza 10 din sirul incepind la **s**, intr-o valoare intreaga binara si asigneaza cei mai semnificativi 8 biti ai acelei valori elementului corespunzator din tabloul de caractere indicat de **array**.

Daca sint **n** valori intregi in sir si mai mult de **n** elemente in tablou, doar primele **n** elemente ale tabloului vor fi valori asignate, iar continutul elementelor ramase va fi nealterat. Daca sint **n** valori intregi in sir si mai putin de **n** elemente in tablou, octetii de dincolo de sfirsitul tabloului vor fi valori asignate ca si cum ar fi elemente ale tabloului iar datele pot fi supracopiate cu erori. Este responsibilitatea programatorului sa previna sau sa asigure cele necesare pentru aceste situatii.

INITW

```
initw(array,s)
     int *array;
     char *s;
```

initw permite o initializare relativ convenabila a tablourilor de intregi. Trebuie trimisi doi parametri: primul, **array**, trebuie sa fie un pointer spre un tablou de intregi, iar al doilea, **s**, trebuie sa fie un pointer spre un sir de caractere ASCII terminat in zero, reprezentind valori de intregi in baza 10, separate de virgule. Cind e apelat, **initb** transforma secvential fiecare valoare intreaga in baza 10 din sirul care incepe la **s**, intr-o valoare intreaga binara si asigneaza acesta valoare elementului corespunzator din tabloul de intregi indicat de **array**.

Daca sint **n** valori intregi in sir si mai mult de **n** elemente in tablou, doar primele **n** elemente ale tabloului vor fi valori asignate, iar continutul elementelor ramase nu va fi alterat. Daca sint **n** valori intregi in sir si mai putin de **n** elemente in tablou, octetii de dincolo de sfirsitul tabloului vor fi valori asignate ca si cum ar fi elemente ale tabloului si datele pot fi supracopiate cu eroare. Este responsabilitatea programatorului sa previna si sa asigure cele necesare pentru aceste situatii.

INP

```
int inp(port)

     char port;
```

inp intoarce valoarea prezenta la portul de intrare desemnat prin **port** dupa executia unei instructiuni IN de masina pentru acel port. (Aceasta functie e disponibila doar la masinile la care instructiunea IN sau una echivalenta are sens.)

ISALNUM

```
int isalnum(c)

    char c;
```

isalnum intoarce adevarat (1) daca **c** e un caracter alfanumeric ASCII, altfel intoarce fals (0).

ISALPHA

```
int isalpha(c)

    char c;
```

isalpha intoarce adevarat (1) daca **c** e un caracter alfabetic ASCII, altfel intoarce fals (0).

ISASCII

```
int isascii(c)

    char c;
```

isascii intoarce adevarat (1) daca **c** este un caracter ASCII, altfel intoarce (0).

ISCNTRL

```
int iscntrl(c)

    char c;
```

iscntrl intoarce adevarat (1) daca **c** este un caracter de control, altfel intoarce (0).

ISDIGIT

```
int isdigit(c)

    char c;
```

isdigit intoarce adevarat (1) daca **c** e un caracter ASCII reprezentind una din cifrele zecimale 0-9, altfel intoarce (0). (Vezi K&R, pag. 127, 156)

ISLOWER

```
int islower(c)

    char c;
```

islower intoarce adevarat (1) daca **c** e un caracter alfabetic ASCII "lower case" (cu litera mica), altfel intoarce (0). (Vezi K&R, pag. 156)

ISNUMERIC

```
int isnumeric(c,radix)

     char c;

     int radix;
```

isnumeric intoarce adevarat (1) daca **c** e un caracter ASCII reprezentind o cifra valida din sistemul de numerare cu baza specificata in **radix**, altfel intoarce (0).

De exemplu:

```
isnumeric('A',15)
```

intoarce adevarat. **isnumeric** e definit doar daca 1 < radix < 36.

ISPRINT

```
int isprint(c)

     char c;
```

isprint intoarce adevarat (1) daca **c** e un caracter ASCII tiparibil, altfel intoarce (0).

ISPUNCT

```
int ispunct(c)

     char c;
```

ispunct intoarce adevarat (1) daca **c** este un caracter ASCII reprezentind un semn de punctuatie, altfel intoarce (0).

ISSPACE

```
int isspace(c)

     char c;
```

isspace intoarce adevarat (1) daca **c** e un caracter ASCII reprezentind un spatiu (blank), un tab sau un linie noua (\n), altfel intoarce un (0). (Aceasta functie e inclusa pentru compatibilitate cu BDS C; functia standard UNIX C e **iswhite**. (Vezi K&R, pag. 156)

ISUPPER

```
int isupper(c)

     char c;
```

isupper intoarce adevarat (1) daca **c** e un caracter alfabetic ASCII "upper case" (cu majuscula), altfel intoarce (0). (Vezi K&R, pag. 145,156)

ISWHITE

```
int iswhite(c)

     char c;
```

iswhite intoarce adevarat (1) daca **c** e un caracter ASCII reprezentind un spatiu, un tab, un linie noua (\n) sau un formfeed (tabulare verticala), altfel intoarce (0).

KBHIT

```
int  kbhit()
```

kbhit testeaza daca s-a tastat ceva la consola -- intorcind adevarat daca da, si fals daca nu. Mai precis, **kbhit** intoarce adevarat (non-zero) daca un caracter e prezent la intrarea standard (CP/M dispozitivul CON: de obicei tastatura consolei), altfel intoarce (0). Aceasta functie nu e disponibila sistemului care nu are o astfel de functie (ca de ex. UNIX).

MOVMEM

```
movmem(source,dest,n)

     char *source, *dest;
     int n;
```

movmem copiaza continutul a **n** octeti consecutivi din memorie, (incepind de la **source**) in **n** octeti consecutivi incepind de la **dest**. Nu exista restrictii in privinta suprapunerii acestor 2 zone. Octetii din zona indicata de **source** nu sint alterati decit daca sint suprascrisi ca rezultat al suprapunerii dintre cele 2 zone.

OPEN

```
FILE *open(fspec,mode)

     FILESPEC *fspec;
     int mode;
```

open deschide fisierul specificat in **fspec** pentru operatii I/E directe (nebufferate). <u>Acest fisier trebuie sa fie creat prin</u> **<u>creat.</u>** **open**, daca reuseste, intoarce un pointer spre un descriptor valid de fisier pentru fisierul specificat. Trebuie sa memorati acest pointer pentru utilizarile ulterioare in program. **open** intoarce ERROR (-1) si nu deschide fisierul daca :

1) nu e disponibila suficienta memorie utilizator pentru un descriptor de fisier nou,

2) specificatia data fisierului e invalida,

3) pointerul specificat fie nu exista, fie nu a fost creat prin **creat**, sau

4) fisierul nu s-a putut deschide din cauza unei erori la nivelul sistemului de operare.

Pentru **mode** trebuie sa specificati: 0, 1, 2, pentru a determina modul I/E al fisierului, conform tabelului:

0	mod doar de citire
1	mod doar de scriere
2	mod de citire - scriere

OTELL

unsigned int otell(fd)

FILE *fd;

otell intoarce offset-ul octetilor de la inceputul blocului (de 512 octeti) curent accesat al fisierului, la care va incepe urmatoarea operatie I/E pe acel fisier. Fisierul e specificat de pointerul **fd** spre descriptorul sau de fisier. **ottel** nu indica in cadrul carui bloc de 512 octeti va incepe operatia I/E. Vezi **rtell** si **tell**.

OUTP

outp(port,b)
char port,b;

outp plaseaza octetul **b** la portul de iesire desemnat de **port** si executa o instructiune de masina OUT pentru acel port. (Aceasta functie e disponibila doar pe masini pentru care instructiunea OUT are sens.)

PAUSE

pause()

pause suspenda executarea programului apelant, pina cind e tastat un caracter la tastatura consolei. **pause** executa o bucla nula si testeaza daca s-a introdus ceva la intrarea standard (CP/M dispozitivul CON: de obicei tastatura consolei), in caz afirmativ iesind din bucla si intorcind controlul la programul apelant. Inainte de activarea **pause** trebuie preluate caracterele de la intrarea standard.

PEEK

char peek(addr)

char *addr;

peek intoarce continutul octetului de memorie de la **addr**. Functia **peek**, care a fost inclusa pentru compatibilitate cu BDS C, e redundanta in C deoarece adresarea indirecta e una din facilitatile acestui limbaj.

PGETC

char pgetc(fd)

FILE *fd;

pgetc e identic cu **getc** cu exceptia ca ori de cite ori intilneste un caracter CR (\r) imediat urmat de un caracter de linie noua (\n), intoarce doar linia noua. **pgetc** transforma astfel linii din fisierele format CP/M in format UNIX.

POKE

```
poke(addr,b)

        char *addr, b;
```

poke scrie octetul **b** in octetul de memorie de la **addr**. Functia **poke**, care a fost inclusa pentru compatibilitate cu BDS C, e redundanta in C deoarece adresarea indirecta e una din facilitatile acestui limbaj.

PPUTC

```
pputc(c,fd)

        char c;
        FILE *fd;
```

pputc e identic cu **putc** cu exceptia ca ori de cite ori e trimis un caracter de linie noua (\n) in **c**, scrie mai intii un caracter CR (\r) in bufferul I/E al fisierului si apoi scrie caracterul de linie noua care a fost trimis. **pputc** transforma liniile scrise in fisierele format UNIX in format CP/M.

PRINTF

```
printf(format,arg1,arg2,...)

        char *format;
        ...
```

printf scrie un sir de iesire formatat la iesirea standard (CP/M dispozitivul CON: de obicei tastatura consolei). Lui **printf** trebuie sa ii fie trimis pointerul, **format**, pentru un sir terminat in zero. (O constanta de sir e deasemenea valida pentru **format** deoarece evalueaza un sir terminat in zero). Acest sir controleaza generarea sirului de iesire. Lui **printf** i se pot trimite o serie de alte argumente: **arg1**, **arg2**, etc. Argumentele individuale in aceasta serie pot fi pointeri intregi, intregi fara semn sau pointeri de siruri. Doar primul argument, **format**, e obligatoriu, toti ceilalti sint optionali.

printf e o functie de listare care trebuie sa fie declarata inainte de utilizare. (Vezi sfirsitul cap. II).

Sirul indicat de **format** poate sa contina caractere binare sau subsiruri speciale incepind cu caracterul **%** , denumite <u>specificatii de conversie</u>. Orice caracter obisnuit intilnit de **printf**, in timp ce scaneaza sirul de la dreapta spre stinga, e scris la iesirea standard. Fiecare specificatie de conversie, cind e intilnita, face ca valoarea urmatorului argument in seria **arg1**, **arg2**, etc, sa fie transformata si formatata dupa cum e specificat si scrisa la iesirea standard.

Dupa caracterul **%**, in fiecare specificatie de conversie poate sa apara:

1) un semn minus optional, '-', care daca exista, face ca valoarea transformata sa fie aliniata la stinga in cimpul sau. Alinierea la dreapta e implicita.

2) un sir optional de cifre in baza 10, care specifica numarul minim de caractere din cimpul in care urmeaza sa fie scrisa voloarea. Valoarea transformata nu va fi niciodata trunchiata. Totusi, daca contine mai putine caractere decit sint specificate, sirul va fi completat la stinga (sau dreapta, daca s-a specificat alinierea spre stinga) cu spatii, la latimea specificata. Daca acest sir de cifre incepe cu zero, valoarea transformata va fi completata cu zero-uri in loc de spatii.

3) un alt sir optional cu cifre in baza 10, care trebuie sa fie precedate de un punct, . , specificind numarul maxim de caractere de copiat dintr-un sir terminat in zero.

4) un caracter, denumit caracter de conversie, indicind tipul de conversie ce trebuie realizat.

Din cele mentionate mai sus, numai caracterul de conversie trebuie sa fie prezent intr-o specificatie de conversie. Toate celelalte, daca exista, trebuie sa fie prezentate in ordinea mentionata mai sus.

Caracterele (valide) de conversie si tipurile de conversie pe care le specifica sint:

-- cel mai putin semnificativ octet al argumentului e interpretat ca un caracter. Acel caracter e scris doar daca e tiparibil.

-- argumentul, care trebuie sa fie un intreg, e transformat in notatie zecimala.

-- argumentul, care trebuie sa fie un intreg, e transformat in notatie octala.

-- argumentul, care trebuie sa fie un intreg, e transformat in notatie hexazecimala.

-- argumentul, care trebuie sa fie un intreg fara semn, e transformat in notatie zecimala.

-- argumentul e interpretat ca un pointer de sir. Caracterele din sirul indicat sint citite si scrise pina cind fie e citit un caracter zero, fie s-a scris un numar maxim specificat optional de caractere. Vezi punctul 3 de mai sus.

-- caracterul % e scris. Aceasta e o secventa "esçape". Nu sint implicate argumente.

(Vezi KGR. pag.7,11,145)

PUTC

```
int putc(c,fd)

      char c;
      FILE *fd;
```

putc scrie caracterul **c** in bufferul I/E asociat unui fisier deschis pentru iesirea la nivel de buffer prin **fopen**, incepind de la locatia curenta in acel buffer. Ori de cite ori acel buffer se umple, el este automat scris (adica intregul sau continut e inclus in fisier). Fisierul e specificat de pointerul **fd** catre descriptorul sau de fisier.

putc intoarce ERROR (-1) si nu scrie caracterul daca:

1) **fd** nu indica un descriptor valid de fisier,

2) fisierul nu a fost deschis pentru o scriere la nivel de buffer prin **fopen**, sau

3) bufferul nu a putut fi scris din cauza unei erori la nivelul sistemului de operare.

Altfel, **putc** scrie caracterul in bufferul I/E al fisierului si intoarce SUCCESS (0). Totusi, din cauza bufferarii operatiilor de I/E ale fisierului, o asemenea valoare de intoarcere nu garanteaza ca acelasi caracter va fi scris cu succes in fisier deoarece erorile care rezulta si afecteaza iesirea unei apelari speciale ale lui **putc** nu poate sa iasa in evidenta pina la un apel ulterior al unei functii care duce la scrierea bufferului in fisier.(Vezi K&R, pag. 152,166)

PUTCHAR

```
putchar(c)

      char c;
```

putchar scrie caracterul **c** la iesirea standard (CP/M dispozitivul CON: de obicei ecranul consolei). (Vezi K&R, pag. 13, 144, 152)

PUTS

```
int puts(s)

     char *s;
```

puts scrie sirul care incepe la **s**, la iesirea standard (CP/M dispozitivul CON: de obicei ecranul consolei). Toate comenzile CR trebuie sa apara explicit in acest sir.

PUTW

```
int putw(i,fd)

     int i;
     FILE *fd;
```

putw scrie intregul **i** in bufferul I/E asociat unui fisier deschis pentru iesirea la nivel de buffer prin **fopen**, incepind de la locatia curenta din acel buffer. Ori de cite ori bufferul se umple, el este automat scris (adica, intregul continut e scris in fisier). Fisierul e specificat de pointerul **fd** spre descriptorul sau.

putw intoarce ERROR (-1) si nu scrie intregul daca:

1) **fd** nu indica un descriptor valid de fisier,

2) fisierul nu a fost deschis pentru iesirea la nivel de buffer prin **fopen**, sau

3) bufferul nu a putut fi scris din cauza unei erori la nivelul sistemului de operare.

Altfel, **putw** scrie intregul in bufferul I/E al fisierului si intoarce SUCCESS (0). Totusi, datorita bufferarii operatiilor I/E ale fisierului, o astfel de valoare de intoarcere nu garanteaza ca acelasi intreg va fi scris cu succes in fisier, deoarece erorile care rezulta si afecteaza iesirea unei apelari la **putc** nu pot fi scoase in evidenta pina la o apelare ulterioara a unei functii care duce la scrierea bufferului in fisier. Apelarile lui **putc** si **putw** pot fi intercalate. Fisierele scrise cu **putc** si **putw** pot fi citite folosind **getc** si **getw**.

RAND

```
int rand()
```

rand intoarce valoarea urmatoare intr-o secventa de numere pseudo-aleatoare initiata printr-o apelare anterioara a lui **srand**. Valorile din secventa se vor inscrie in gama 0 pina la 65535.

Expresia C:

```
     rand() % n
```

va evalua un intreg mai mare sau egal cu 0, dar mai mic decit n.

READ

```
int read(fd,bufr,n)

     FILE *fd;
     char *bufr;
     int n;
```

read citeste maximum **n** octeti dintr-un fisier deschis pentru intrarea directa sau la nivel de buffer, incepind de la locatia curenta a pointerului I/E al fisierului, in bufferul de memorie indicat de **bufr**. Fisierul e specificat de pointerul **fd** spre descriptorul sau de fisier.

Trebuie sa definiti bufferul indicat de **bufr** astfel incit sa contina cel putin n octeti . n trebuie sa fie un multiplu intreg pozitiv al marimii inregistrarii de sistem. (O inregistrare de sistem este unitatea minima de date transferate in timpul operatiilor I/E ale fisierului. Sub CP/M o inregistrare de sistem e de 128 octeti. Consultati documentatia sistemului pentru informatii suplimentare.)

Pointerul I/E al fisierului va indica intotdeauna spre inceputul unei inregistrari de sistem. Dupa apelarea lui **read**, pointerul I/E va indica inceputul urmatoarei inregistrari dupa ultima citita.

Daca nu au loc erori, **read** intoarce numarul real al octetilor cititi. Daca acesti octeti sint cititi dintr-un fisier, **read** intoarce fie un multiplu, al marimii inregistrarii de sistem, fie zero. Zero va fi intors doar daca s-a ajuns la sfirsitul fisierului. Daca octetii sint cititi dintr-un dispozitiv serial (ca de ex. CP/M dispozitiv CON: sau RDR:) deschis ca un fisier, **read** intoarce (1), deoarece doar un singur octet pe apelare la **read** poate fi citit dintr-un dispozitiv serial. **read** intoarce ERROR (-1) si nu incearca sa citeasca fisierul daca:

1) fisierul nu a fost deschis pentru intrare,
2) **n** e mai mic decit marimea inregistrarii de sistem, sau
3) fisierul nu a putut fi citit din cauza unei erori la nivel de sistem de operare. (Vezi K&R, pag. 160)

RENAME

```
int rename(fname,fspec)

     char *fname;
     FILESPEC *fspec;
```

rename redenumeste fisierul specificat in **fspec**, cu numele continut in sirul terminat in zero indicat de **fname**. (O constanta de sir, ca de ex. "newname " este deasemenea valida pentru **fname** deoarece evalueaza un pointer spre un sir terminat in zero.) Daca exista, sint neschimbate numele si numarul discului.

RESET

```
reset(n)

     int n;
```

reset determina ca executia programului sa se intoarca la un punct setat de o apelare anterioara a lui **setexit**. Acest transfer are infatisarea unei intoarceri de la **setexit**. Parametrul **n** trimis lui **reset** apare ca valoarea intoarsa de **setexit**.

reset si **setexit** impreuna permit o codificare mai simpla si mai sigura a iesirilor repetate la un punct comun, in special atunci cind transferurile reclama lamurirea unui numar de nivele de apelari de functie. De exemplu, la scrierea unui editor interactiv puteti apela **setexit** in virful buclei de comanda

si puteti testa daca valoarea sa de intoarcere aparenta a fost sau nu egala cu zero (0). Fiecare valoare non-zero poate fi folosita pentru a indica o conditie diferita de eroare. Numarul de eroare poate fi tiparit si executarea buclei de comanda poate continua. Apelarile la **reset** vor fi distribuite in locuri adecvate pe lungimea buclei. In fiecare caz, parametrul trimis la **reset** va indica prezenta (non-zero) sau absenta (zero) unei conditii speciale de eroare.

reset si **setexit**, desi seamana a functii ca sintaxa si utilizare, sint implementate ca directive de preprocesare a compilatorului si nu ca functii. De aceea nu le veti gasi in nici un fisier de functii de biblioteca standard.

RINDEX

char *rindex(s,c)

char *s, c;

rindex intoarce un pointer la ultima aparitie a caracterului **c** din sirul care incepe la **s**. **index** intoarce o valoare de pointer nula daca **c** nu apare in sir.

RTELL

unsigned int rtell(fd)

FILE *fd;

rtell intoarce (in blocuri de 512 octeti) offsetul -fata de inceputul fisierului -blocului de 512-octeti al fisierului in cadrul caruia va incepe urmatoarea operatie de I/E. **rtell** nu indica offset-ul in acel bloc in care va incepe operatia I/E. Fisierul e specificat de pointerul **fd** spre descriptorul sau de fisier. Vezi **otell** si **tell**.

SBRK

char *sbrk(n)

int n;

sbrk adauga **n** octeti la memoria utilizator (incrementeaza CCEDATA prin **n**). Daca reuseste, **sbrk** intoarce un pointer la primul octet din blocul adaugat. **sbrk** intoarce -1 si nu adauga nici un octet la memoria utilizator daca un bloc de marimea specificata se va :

1) suprapune peste un program memorat,
2) suprapune peste stiva de executie, sau
3) depaseste memoria utilizator disponibila.

IMPORTANT: Apelarea lui **sbrk** cu un argument negativ, nu trebuie sa aiba loc intre apelarile lui **alloc** (vezi K&R, pag. 157).

SCANF

```
int scanf(format,arg1,arg2,...)

    char *format;
    ...
```

scanf citeste un sir de intrare formatat de la intrarea standard (CP/M dispozitivul CON: de obicei tastatura consolei). Sub controlul sirului format indicat de primul argument, **format**, **scanf** extrage o serie de subsiruri, <u>cimpuri de intrare</u>, din sirul sau de intrare, transforma valorile reprezentate in fiecare din aceste cimpuri, <u>valori de intrare</u>, si asigneaza secvential aceste valori transformate obiectelor indicate de argumentele ramase **arg1**, **arg2**, ...

scanf e o functie de listare care trebuie declarata inainte de utilizare. Vezi "Declararea functiilor de listare" de la sfirsitul cap.II.

Ca prim argument, lui **scanf** trebuie sa-i fie trimis un pointer, **format**, la un sir **format** terminat in zero. (O constanta de sir e deasemenea valida pentru **format** deoarece evalueaza un pointer spre un sir terminat in zero). O serie de argumente, **arg1**, **arg2**, ..., pot fi trimise la **scanf**; toate trebuie sa fie pointeri. Obiectele individuale indicate de **arg1**, **arg2**, ... pot fi caractere, tablouri de caractere sau intregi.

Sirul **format** poate sa contina caractere de despartire (adica spatiu, tab si linie noua), caractere obisnuite sau subsiruri speciale care incep cu **%** , denumite <u>specificatii de conversie</u>. Prima specificatie de conversie din sirul **format** corespunde si determina limitele primului cimp de intrare din sirul de intrare. De asemenea determina si tipul de conversie care urmeaza a fi efectuat pe valoarea de intrare reprezentata in acel cimp. Fiecare pereche succesiva de specificatii de conversie si cimpuri de intrare se afla in aceeasi relatie.

In urma unui caracter %, in fiecare specificatie de conversie pot apare:

1) un caracter optional de suprimare a asignarii, '*', care, daca exista, provoaca salt peste cimpul de intrare corespunzator.

2) un sir optional de cifre in baza 10 specificind numarul maxim de caractere in cimpul corespunzator de intrare.

3) un caracter, numit <u>caracter de conversie</u>, indicind tipul de conversie care urmeaza a fi efectuat pe valoarea corespunzatoare de intrare.

Din cele de mai sus, doar caracterul de conversie trebuie sa fie prezent intr-o specificatie de conversie. Celelalte caractere, daca exista, trebuie sa fie ordonate conform listarii de mai sus.

Caracterele valide de conversie si tipurile de conversie sint:

-- un singur caracter % e asteptat la acest moment in sirul de intrare. Aceasta este o secventa "escape" - nu are loc nici o asignare.

-- valoarea de intrare e interpretata ca un caracter. Argumentul corespunzator trebuie sa fie un pointer de caracter. Saltul normal peste caracterele de spatiu este suprimat. Folositi **%1s** pentru a citi urmatorul caracter diferit de "spatiu".

Daca e mentionata si latimea unui cimp, argumentul corespunzator trebuie sa fie un pointer spre un tablou de caractere si va fi citit numarul specificat de caractere.

-- valoarea de intrare e interpretata ca un sir de caractere. Argumentul corespunzator trebuie sa fie un pointer spre un tablou de caractere destul de mare pentru a cuprinde sirul, plus un caracter terminal nul adaugat de **scanf**. Cimpul de intrare e terminat printr-un spatiu sau linie noua, sau atunci cind numarul maxim de caractere a fost citit.

-- valoarea de intrare e interpretata ca un sir de caractere. Argumentul corespunzator trebuie sa fie un pointer spre un tablou de caractere destul de mare pentru a cuprinde sirul plus un caracter terminal nul adaugat de **scanf**.

Locul unde se termina cimpul de intrare e determinat in modul urmator:

Paranteza stinga de mai sus e urmata de un set de caractere si o paranteza dreapta. Daca primul caracter in acel set nu e un accent circumflex, '^', cimpul de intrare e terminat de primul caracter care nu se afla in setul dintre

paranteze. Daca primul caracter e un accent circumflex, cimpul de intrare e terminat de primul caracter din set in cadrul parantezelor (semnul '^' e exclus).

-- valoarea de intrare e interpretata ca un intreg in baza zece si e transformata intr-un intreg binar. Argumentul corespunzator trebuie sa fie un pointer intreg.

-- valoarea de intrare e interpretata ca un intreg in baza opt si e transformata intr-un intreg binar. Argumentul corespunzator trebuie sa fie un pointer intreg.

-- valoarea de intrare e interpretata ca un intreg in baza 16 si e transformata intr-un intreg binar. Argumentul corespunzator trebuie sa fie un pointer intreg.

Sarcina principala al lui **scanf** e de a determina granitele cimpului de intrare in sirul sau de intrare care contine valorile de intrare care urmeaza a fi transformate si asignate. Pentru a afla aceste subsiruri, **scanf** scaneaza caracterele din sirul sau de intrare, comparind fiecare dintre ele cu caracterele corespunzatoare din sirul indicat de **format**. Daca un caracter din sirul de intrare se potriveste cu caracterul coresunzator din sirul **format**, acesta este respins si se citeste urmatorul caracter din sirul de intrare. Daca insa caracterele corespunzatoare nu se potrivesc, **scanf** se intoarce imediat. Urmariti ca orice spatiu de despartire din sirul de intrare sa se potriveasca cu orice spatiu de despartire din sirul **format**. Spatiul de despartire din sirul **format** e optional (ignorat) in timp ce in sirul de intrare poate sa delimiteze cimpurile de intrare. Astfel, caracterele corespunzatoare nu sint doar acele caractere care au acelasi numar de octeti de la inceputul sirurilor respective (acelasi numar de ordine in sir).

Ori de cite ori e intilnit in sirul **format** caracterul % care introduce o specificatie de conversie, caracterul corespunzator din sirul de intrare e presupus a fi primul octet dintr-un cimp de intrare. Un cimp de intrare se extinde fie pina cind e intilnit un caracter de spatiu in sirul de intrare, fie pina cind s-a citit numarul de octeti specificati pentru latimea cimpului.

Caracterele de conversie **c** si **[** sint singurele exceptii ale acestei reguli dealtfel generale. Orice caracter nepotrivit intr-un cimp de intrare face ca **scanf** sa se intoarca imediat.

scanf intoarce fie numarul valorilor de intrare transformate pe care le-a asignat, fie constanta EOF daca nu exista nici o intrare la intrarea standard. (vezi K&R, pag. 147)

SEEK

```
int seek(fd,offset,origin)

     FILE *fd;
     int offset;
     int origin;
```

seek seteaza valoarea pointerului I/E al fisierului asociat cu un fisier deschis. Fisierul e specificat de pointerul **fd** catre descriptorul sau de fisier si poate sa fi fost deschis fie pentru I/E directa fie la nivel de buffer.

seek e folosit in primul rind in legatura cu **tell** si functiile directe I/E ale fisierului **read** si **write**. **seek** trebuie sa fie folosit in legatura cu functiile de I/E la nivel de buffer ale fisierului pentru a preveni pierderea de date.

Valoarea asignata **offset**-ului are o interpretare diferita, functie de valoarea asignata lui **origin**:

Daca **origin** = 0, atunci pointerul fisierului I/E va indica spre inceputul fisierului + octetii **offset**.

Daca **origin** = 1, atunci pointerul fisierului I/E va indica pozitia sa curenta din fisier + octetii **offset**.

Daca **origin** = 2, atunci pointerul fisierului I/E va indica spre sfirsitul

fisierului + octetii **offset**.

Daca **origin** = 3, atunci pointerul fisierului I/E va indica spre inceputul fisierului + 512 x octetii **offset**.

Daca **origin** = 4, atunci pointerul fisierului I/E va indica spre pozitia sa curenta din fisier + 512 x octetii **offset**.

Daca **origin** = 5, atunci pointerul fisierului I/E va indica spre sfirsitul fisierului + 512 octetii **offset**.

(vezi K&R, pag. 164, 169)

SETEXIT

int setexit()

setexit seteaza locatia sa ca punct de reset -- punctul la care apelarile consecutive spre **reset** transfera executia programului. Fiecare apelare a lui **reset** care urmeaza provoaca o intoarcere aparenta de la functia **setexit**.

setexit pare sa intoarca valoarea parametrului ,**n**, care a fost trecut la **reset**. Vezi **reset**.

SETMEM

setmem(p,n,b)

```
char *p;
int n;
char b;
```

setmem seteaza **n** octeti consecutivi incepind de la **p** cu valoarea specificata in **b**. Puteti folosi **setmem** pentru a initializa o varietate de buffere si tablouri.

SLEEP

sleep()

sleep suspenda executia pentru **n** zecimi de secunda la un CPU Z80 care ruleaza la 4MHz. Puteti ajusta aceasta functie la un CPU si/sau ceas diferit, schimbind valoarea uneia sau a doua constante.

SPRINTF

sprintf(s, format, arg1, arg2,...)

```
char *s, *format;
...
```

sprintf e identic cu **printf** cu exceptia ca scrie iesirea formatata in sirul care incepe la **s**, spre deosebire de **printf**, care scrie iesirea in iesirea standard, si **fprintf** care scrie iesirea intr-un fisier. **sprintf** adauga un caracter zero la sirul de iesire formatat (vezi K&R, pag. 150). **sprintf** e o functie de listare.

SRAND

srand(seed)

```
int seed;
```

srand initializeaza valoarea de intoarcere a lui **rand** la valoarea trecuta in **seed**.

SSCANF

```
int sscanf(s,format,arg1,arg2,...)

        char *s, *format, *arg1, arg2, ...;
        ...
```

sscanf e identic cu **scanf** (si **fscanf**) cu exceptia ca sirul sau de intrare formatat e citit din sirul terminat in zero incepind la **s** si nu din intrarea standard. **sscanf** nu citeste caracterul terminal nul (vezi K&R, pag. 150). **sscanf** e o functie de listare.

STRCAT

```
char *strcat(s1,s2)

    char *s1, *s2;
```

strcat adauga o copie a sirului incepind la **s2** la sfirsitul sirului incepind la **s1**, creind astfel un singur sir terminat in zero. Notati ca sirul rezultat incepe la **s1** si contine un singur caracter terminal nul.

strcat intoarce un pointer spre sirul rezultat, identic cu parametrul **s1** care a fost trimis (vezi K&R, pag. 44).

STRCMP

```
int strcmp(s1,s2)

    char *s1, *s2;
```

strcmp compara sirul incepind la **s1** cu sirul incepind la **s2**. Aceasta comparatie e similara cu o comparatie alfabetica cu exceptia ca se bazeaza pe valorile numerice ale caracterelor corespunzatoare din cele doua siruri. Aceasta comparatie se incheie atunci cind e intilnit primul caracter zero in oricare sir.

strcmp intoarce un intreg pozitiv, zero (0) sau un intreg negativ daca sirul incepind la **s1** e mai mare , egal sau respectiv mai mic decit sirul incepind la **s2** (vezi K&R, pag. 101).

STRCPY

```
strcpy(s1,s2)

    char *s1, *s2;
```

strcpy copiaza sirul care incepe la **s2** in sirul incepind la **s1**, oprindu-se dupa un caracter zero care a fost copiat. Daca lungimea sirului incepind la **s2** e mai mare decit lungimea celui incepind la **s1**, datele din octetii care depasesc cel de a-l doilea sir pot fi supracopiate cu eroare.

strcpy intoarce parametrul **s1**, care i-a fost trimis (vezi K&R, pag 100).

STREQ

```
int *streq(s1,s2)

     char *s1, *s2;
```

streq compara primele **n** caractere din sirurile care incep la **s1** si **s2**, unde **n** e numarul de caractere (exclusiv zero-ul terminal) din sirul incepind la **s2**. **streq** intoarce **n** in cazul in care caracterele corespunzatoare din cele doua siruri sint identice; altfel intoarce zero (0).

STRLEN

```
int strlen(s)

     char *(s);
```

strlen intoarce numarul de caractere (exclusiv zero-ul terminal) din sirul incepind la **s** (vezi K&R, pag. 36, 95, 98).

STRNCAT

```
char *strncat(s1,s2,n)

     char *s1, *s2;
     int n;
```

strncat e identic cu **strcat** cu exceptia ca **strncat** adauga cel mult **n** caractere din sirul incepind la **s2** (trunchiind din dreapta) la sfirsitul sirului care incepe la **s1**.

STRNCPY

```
char *strncpy(s1,s2,n)

     char *s1, *s2;
     int n;
```

strncpy e identic cu **strcpy** cu exceptia ca **strncpy** copiaza exact **n** caractere in sirul care incepe la **s1**, cu trunchiere sau completare cu zero a sirului incepind la **s2** daca e necesar. Sirul rezultant poate sa nu se termine in zero daca sirul incepind la **s2** contine **n** sau mai multe caractere.

TELL

```
unsigned int tell(fd)

     FILE *fd;
```

tell intoarce offset-ul in octeti de la inceputul unui fisier pina unde va incepe urmatoarea operatie I/E pe acel fisier. Fisierul e specificat de pointerul **fd** spre descriptorul sau de fisier. Daca **tell** e apelat pentru un fisier mai mare de 64K, valoarea lui de intoarcere e supusa depasirii aritmetice (overflow). Vezi **otell** si **rtell**.

TOLOWER

```
int tolower(c)

     char c;
```

tolower intoarce un echivalent cu litere mici a lui **c** daca **c** e un caracter alfabetic ASCII cu majuscule, altfel intoarce **c** (Vezi K&R, pag. 145, 156).

TOPOFMEM

```
char *topofmem()
```

topofmem intoarce CCEDATA (vezi **brk**, **evnbrk** si **sbrk**).

TOUPPER

```
int toupper(c)

     char c;
```

toupper intoarce echivalentul cu majuscula al lui **c** daca **c** e un caracter alfabetic ASCII cu litere mici, altfel intoarce **c** (vezi K&R, pag. 156).

UNGETC

```
ungetc(c,fd)

     char c;
     FILE *fd;
```

ungetc scrie caracterul **c** in cel mai recent citit octet al bufferului I/E asociat unui fisier deschis pentru o intrare la nivel de buffer prin **fopen**. **ungetc** decrementeaza si pointerul spre urmatorul octet de citit din bufferul I/E al fisierului, astfel incit indica spre octetul care tocmai a fost scris.

ungetc, daca reuseste, intoarce o valoare nedefinita. **ungetc** intoarce ERROR (-1) daca nu si-a putut realiza functia. De exemplu, daca fisierul specificat nu a fost deschis pentru intrarea la nivel de buffer prin **fopen**.

Apelarea lui **ungetc** pentru un fisier nu serveste la nimic daca nu au fost apelate anterior pentru acelasi fisier **fgets**, **fscanf**, **getc** sau **getw** (functii de intrare ale fisierului bufferat). Realizarea scopului propus se poate garanta doar pentru o singura apelare a lui **ungetc** intre apelarile facute la functiile de intrare ale fisierului bufferat (vezi K&R, pag. 156).

UGETCHAR

```
ugetchar(c)

     char c;
```

ugetchar face ca apelarea urmatoare a lui **getchar** sa intoarca pe **c**. Apelind mai mult de o data pe **ugetchar** intre apelarile succesive ale lui **getchar** nu vor avea nici un efect asupra starii intrarii standard.

UNLINK

```
int unlink(fspec)

        FILESPEC *fspec;
```

unlink sterge fisierul specificat in **fspec** din sistemul de fisiere. **unlink** intoarce SUCCESS (0) daca fisierul a fost sters cu succes. **unlink** intoarce ERROR (-1) si nu sterge fisierul daca:

1) specificatia data fisierului nu e valida sau
2) fisierul nu a putut fi sters din cauza unei erori la nivelul sistemului de operare (vezi K&R, pag. 163).

WRITE

```
int write(fd,buffer,num-bytes)

        FILE *fd;
        char *buffer;
        int num-bytes;
```

write scrie numarul de octeti specificati in **num-bytes** din bufferul indicat de **buffer** intr-un fisier deschis pentru iesirea directa (nebufferata). Fisierul e specificat de un pointer, **fd**, spre descriptorul sau de fisier.

Valoarea asignata lui **num-bytes** trebuie sa fie cel putin egala cu numarul de octeti dintr-o inregistrare de sistem. Sub CP/M, o inregistrare de sistem contine 128 octeti; la alte sisteme variaza acest numar. Trebuie sa va asigurati ca bufferul indicat de **buffer** contine cel putin numarul de octeti specificat in **num-bytes**. **write** nu testeaza bufferul.

write intoarce numarul real de octeti scrisi, care va fi intotdeauna un multiplu al marimii inregistrarii de sistem datorita modului de lucru in blocuri cu discul fizic. Daca numarul de octeti pe care il specificati pentru a fi scrisi nu e un multiplu al marimii inregistrarii de sistem, o parte din ultima inregistrare scrisa va contine octeti al caror continut nu l-ati specificat. Daca valoarea intoarsa de **write** e mai mica decit numarul de octeti specificati in **num-bytes**, atunci e foarte probabil ca ati epuizat spatiul pe disc. Acest lucru trebuie tratat ca eroare. Daca descriptorul fisierului e invalid sau fisierul nu poate fi citit, e intoarsa o valoare (-1) pentru a indica o eroare.

Fiecare descriptor de fisier contine un pointer spre urmatoarea inregistrare care urmeaza a fi abordata in operatiile I/E ale fisierului. O apelare a lui **write** avanseaza acel pointer cu numarul de octeti scrisi, chiar daca nu ati specificat continutul tuturor octetilor din ultima inregistrare scrisa. O apelare ulterioara a lui **read** sau **write** va incepe in pozitia noua a acestui pointer. De aceea, scrierea unui numar de octeti care nu sint multipli ai marimii inregistrarii de sistem poate lasa date nespecificate (deseuri) in fisier. Apelarea lui **seek** poate altera pozitia acestui pointer I/E de fisier fara citire sau scriere, insa totusi va continua sa indice spre inceputul unei inregistrari (vezi K&R, pag. 160).

CAPITOLUL IV

Inserarea rutinelor in biblioteca de executie C2.RH

Limbajul C permite utilizatorului sa deplaseze codul de operare in fisierul C2.RH al optimizatorului. Acesta e fisierul care e plasat in mod automat la inceputul unui program C si asigura rutinele necesare de executie. Avantajul e acela ca rutinele operationale nu trebuie sa fie recompilate si reoptimizate de fiecare data cind programul e recompilat.

Pentru realizarea acestui lucru, utilizatorul compileaza rutinele necesare, obtinind un fisier in limbaj de asamblare. Acest fisier e concatenat apoi cu fisierul C2.RH. Din acest moment, acele rutine vor fi disponibile fara alte prelucrari ulterioare.

In continuare e prezentata, in etape, descrierea modului in care fisierul CRUNT2.C de subrutine poate fi mutat in rutinele de executie, eliminindu-se astfel nevoia de a folosi o directiva #include "crunt2.c".

1) sinteti pregatit pentru a compila fisierul "CRUNT2.C". Emiteti urmatoarele comenzi:

```
A>CC CRUNT2.C

.
.
.
A>C2 CRUNT2.COD +ASM +ZA -RH -RT
```

Optiunile "+ZA" indica translatorului sa inceapa toate etichetele cu litera "A". Acest lucru evita orice conflict cu codul produs de compilarile normale care fac sa inceapa fiecare eticheta cu un "C". Optiunile -RT si -RH previn includerea fisierelor header si trailer.

2) Apoi, REName fisierul cu CRUNT2.LIB, in pregatirea pentru concatenarea cu fisierul C2.RH.

3) Inainte ca fisierul AUXFN.LIB sa poata fi inclus e necesara o mica operatie de editare. Tastati:

```
A>ED CRUNT2.LIB
*#a
*0p
```

vedeti apoi:

```
;C Optimizer V1.1
    ORG       256
;C Compiler V1.1
 iswhite:
        lxi     h,2
        dad     sp
        mov     l,m
        mvi     h,0
        mov     a,h
        ora     l
        jz      A2
        lxi     h,2
        dad     sp
        mov     l,m
```

```
        mvi     h,0
        lxi     d,-32
        dad     d
        mov     a,h
        ora     l
        jz      A2
        lxi     h,2
        mov     l,e
```

Cind listati pe ecran inceputul fisierului, fisierul trebuie sa fie acelasi (cu exceptia numerelor de versiune) cu cel listat mai sus. Trebuie sa stergeti urmatoarea linie:

```
    ORG     256
```

Apoi tastati "e" pentru a salva fisierul editat.

4) Acum, trebuie sa editati fisierul C2.RH. Pe ecranul monitorului veti vedea:

```
A>ED C2.RH
*#A
*4410p
CCSTART:
;
;       wich CPU ?
        mvi  a,2
        inr  a
        jpe  c$is8080
;       is Z80  -- this code will not work in ROM
cldir   equ  0b0edh   ; ldir instruction
        lxi  h,cldir
        shld cldir1
        shld cldir2

        xra  a
        sta  cldir1+2
        sta  cldir2+2

c$is$8080:
        lxi  d,ccidstr
        MVI  C,9
        CALL CCBDOS
        LHLD 6
        SPHL
        call main
        rst  0

exit    equ  0 .      ; exit()  -- for C
ccidstr:
        db  'SuperSoft C Copyright 1981'
        db  0dh, 0ah, 'Run Time Package V2.3',0DH,0AH,'$'.

; ***** insert any user code here *****

cc0err:
        pop  d
        push b
        mvi  c,9
        call ccbdos
```

```
lxi  h,0
pop  b
ret
```

Avansati pointerul editorului curent deasupra liniei urmatoare:

```
; ***** insert any user code here *****
```

Emiteti urmatoarea comanda:

```
* rcrunt2
```

Acest lucru va face ca editorul sa citeasca fisierul "CRUNT2.LIB"

5) Iesiti din editor cu un "e". In acest moment aveti toate functiile continute in AUXFN.N, in fisierul de executie al sistemului numit C2.RH. Inseamna ca nu va trebui sa mai folositi directiva #INCLUDE "CRUNT2.C". Totusi, vreti - nu vreti, veti compila intotdeauna si aceste functiuni. O optiune excelenta e aceea de a avea disponibile citeva fisiere C2.RH diferite, cu diverse rutine, care va vor folosi la anumite tipuri de programe.

Amintiti-va:

6) Orice rutina poate fi mutata in biblioteca de executie a sistemului. Totusi, nu puteti muta un "main()".

ANEXA A

Diferentele dintre limbajele C SuperSoft si C Standard

Caracteristicile limbajului de C Standard care nu sint inca implementate sint:

- clasa de memorare STATIC,
- declaratii TYPEDEF,
- tipuri de date LONG, FLOAT si DOUBLE,
- declararea si utilizarea cimpurilor de biti,
- initializari.

Toate argumentele formale la o functie trebuie sa fie declarate in cadrul acelei functii. Adica:

```
func(aa, bb, cc)
     int bb;
     int aa, cc;
    {
    ... e acceptat, dar

     func(aa, bb, cc)
    {
    ... va genea un mesaj de prevenire.
```

Generatorul de cod (óptimizorul) nu adauga inca nici un prefix la numele de variabile. Astfel identificatorii globali din codul sursa C pot intra in conflict cu numele si cuvintele cheie ale asamblorului. De aceea trebuie sa evitati utilizarea in codul Dv. sursa C a oricaror simboluri sau cuvinte cheie ale asamblorului. Aceste doua dezavantaje pot fi ulterior remediate.

Nu sint inca implementate directivele parametrizate de preprocesor **#if**, **#ifdef**, **#ifndef**, **#undef**, **#else** si **#endif**.

ANEXA B

Configuratiile disponibile in mod curent

Sistemele curente de operare sint : CP/M-80, MP/M-80, CP/M-86 (si sistemele compatibile); PC-DOS (si sistemele compatibile); UNIX, XENIX (si sistemele compatibile) si Central Data ZMOS.

CPU-uri curente: Intel 8080, Intel 8085, Intel 8086, Intel 186, Intel 188, Intel 286, Zilog Z80, Zilog Z8001, Zilog Z8002 si Zilog Z8003.

SuperSoft furnizeaza orice combinatie valida de sistem de operare si CPU.

ANEXA C

Unele probleme uzuale si solutii

Problema	Solutia
In timpul fazei de asamblare unele functii sint nedefinite	Amintiti-va sa #includeti fisierele care contin functiile necesare.
In timpul fazei de asamblare unele variabile sint nedefinite.	Amintiti-va sa declarati toate variabilele folosite.
In timpul fazei de asamblare unele functii sau variabile sint semnalizate ca erori de dubla definitie.	Accidental, o functie sau o variabila sint definite de doua ori.
Cind folositi I/E pe disc, programul Dv. face lucruri ciudate.	Amintiti-va ca trebuie sa initializati "allocs" la zero (adica allocs = 0.
Cind folositi alocarea, programul face lucruri stranii.	Amintiti-va sa setati allocs la zero (vezi mai sus).
In timpul asamblarii este semnalata o eroare "P".	O variabila sau functie a fost definita de mai multe ori.

ANEXA D

Locatiile functiilor furnizate

Functiile care sint utilizate in ALLOC.C:

alloc	evnbrk	free	

Functiile localizate in C2.RH:

bios	brk	ccall	ccalla
exit	reset	setexit	streq

Functiile localizate in CRUNT2.C:

getchar	gets	isalpha	isdigit
islower	isupper	iswhite	movmem
outchar	puts	sbrk	setmem
strcpy	strlen	toupper	ungetchar

Functiile care sint localizate in FUNC.C:

abs	atoi	bios	getval
index	initb	initw	isalnum
isascii	iscntrl	isnumeric	isprint
ispunct	isspace	kbhit	pause
peek	poke	qsort	rand
rindex	sleep	srand	strcat
strcmp	strncat	strncpy	tolower

Functiile care sint localizate in STDIO.C

lose	creat	exec	execl
abort	folose	fflush	fopen
getc	getw	open	otell
pgetc	pputc	putc	putw
read	seek	putc	putw
unlink	write	tell	ungetc

Functiile care sint localizate in FORMATIO.C

printf	fputs	fscanf	printf
scanf	sprintf	sscanf	

I.P."Filaret" A.III. Dinu Vintilă nr.4
Cd.363/1986/tiraj 1000